SIMetrix/SIMPLIS

Advanced Power System Simulation

SIMPLIS Reference Manual

Copyright ©1992-2007 Catena Software Ltd.

Contact
Catena Software Ltd., Terence House, 24 London Road,
Thatcham, RG18 4LQ, United Kingdom

Tel: +44 1635 866395
Fax: +44 1635 868322
Email: info@catena.uk.com
Internet http://www.catena.uk.com

SIMPLIS is a product of Transim Technology Corporation

Transim
433 NW 4th Avenue, Suite 200 Portland, OR, USA 97209
Phone: 503.450.1355 Fax: 503.295.2720
info@transim.com http://www.transim.com

SIMPLIS

Catena Software Ltd. is a member of the Catena group of companies. See http://www.catena.nl

Table of Contents

Chapter 1 Introduction

Overview....8
Organization of this User Manual....8

Chapter 2 Input File Organization

Overview....10
General Rules for the Input File....10
- Statements and Continuation of Statements....10
- Blank Characters....10
- Blank Lines....10
- Comment Statements....10
- In-line Comments....11
- Continuation of Statements....11
- Device Names....11
- Model Names and Subcircuit Names....12
- Uppercase vs. Lowercase....13
- Integer Entries....13
- Floating-point Entries....14
- Units....16
- Length of Fields and Lines....16

Organization of the Input File....17
- General Circuit....17
- Main Circuit....17
- Subcircuit....18
- General Statements....18

Chapter 3 Device Statements

Overview....20
- Device Statement Format....20
- Node Names....20
- Voltage and Current Polarity Conventions....21
- Parameter Assignments....21
- Controlling Devices....22

SIMPLIS Device Types....22
- Linear Resistors....22
- Linear Inductors and Capacitors....23
- Independent Voltage and Current Sources....23
- Triangular Sources....26

Square Wave Sources 28
Pulse Sources with Zero Rise and Fall Times 30
Sinusoidal Sources 32
Cosinusoidal Source 34
Aperiodic Exponential Pulse Sources 36
Aperiodic Piecewise-Linear Sources..... 38
Mutual Inductances 40
Linear Voltage-Controlled Sources 41
Linear Current-Controlled Sources 42
Ideal Transformers 43
Simple Switches 43
Simple Transistor Switches 44
Piecewise Linear Resistors 45
Piecewise-Linear Inductors and Capacitors 46
Simple Logic Gates 47
Subcircuit Calls / Instantiation 48

Chapter 4 Model Statements

Overview 50
Device Models Used in Simplis 52
Piecewise-Linear Resistor Models 52
Piecewise-Linear Inductor and Capacitor Models 55
Simple Switch Models 57
Simple Transistor Switch Models 59
Simple Logic Device Models 64

Chapter 5 Subcircuit Definition

Overview 86
Subcircuit Definition..... 87
Parent and Child Relationships for Subcircuits 87
.SUBCKT Statement 88
.ENDS Statement (End of Subcircuit Statement) 88
Scope of Definition 90
The Scope of Definition for the Main Circuit..... 90
The Scope of Definition for a Subcircuit..... 90
Scope of Definition for a Device and for a Node 90
External and Local Nodes 91
Subcircuit Calls/Instantiation 92

Chapter 6 Control Statements

Overview 95

Option Statements 95
Control Statements for Setting Initial Conditions 99
Linear and PWL Capacitors 99
Linear and PWL Inductors 99
Setting of Initial States for S and Q Switches 100
Setting of Initial Segment for PWL Resistors 100
Setting of Initial State for Simple Logic Gates 100
Initial Conditions for Devices in a Subcircuit 100
Control Statements for Printing Variables 101
Mapping Names to Node Numbers 103
Creating SIMetrix Plots 104
Control Statements Associated with Analyses 104
.TRAN - Time-Domain Transient Analysis 104
.POP - Periodic Operating Point Analysis 105
.AC - Frequency Domain Analysis 105

Chapter 7 Running SIMPLIS

Overview 106
Running SIMPLIS on a SIMetrix Schematic 106
Adding Extra Netlist Lines 106
Running SIMPLIS for an External Netlist 107
Running SIMPLIS from a Script 107
Running SIMPLIS from a DOS Prompt 108
SIMPLIS Execution 108
Aborting a SIMPLIS Run 109
Automatic Program Suspension by SIMPLIS 109
Netlist Preprocessor 109
Overview 109
Launching Preprocessor 110
Library Search 110
Parameters 111
Passing Parameters to Subcircuits 111
Conditional Lines 111
Looping 112
Running Monte Carlo and Multi-step Analyses 112

Chapter 8 Simplis Data Files

Overview 113
The Listing Data File 113
Error Message Data File 113
The "State of Exit" Data File 113
Switching Instance Data File 114

Time-domain Data Output 114
The Topology Information File 115
Taking Advantage of Existing Files 115
Switching Instance Data File for the POP Analysis 116
Data for the Periodic Operating Point Analysis 116
Print/Plot File for Frequency-Domain Analysis 116

Chapter 9 Simplis-TX Examples

Overview 117
Example 1 -- Rectifier with RC load 117
Example 2 -- 3-Phase Rectifier with Resistive Load 119
Example 3 -- Operational Amplifier with Saturation 121
Example 4 -- Unregulated Converter 124
Example 5 -- Regulated Converter 126
Example 6 -- Saturable Inductor 129
Example 7 -- SCR with RL Load 131

Chapter 10 Simplis-POP

Overview 135
Statements Relating to POP Analysis 136
.POP Statement for POP Analysis 136
Options Associated with POP Analysis 142
Synopsis of the Periodic Operating Point Analysis 143
The Time Variable During POP Analysis 146
How POP Deals with Time Varying Sources 146
The POP_SHOWDATA option for POP Analysis ... 149
Example of Applying the POP Analysis Tool 151

Chapter 11 Simplis-FX

Overview 155
Statements Relating AC Analysis 157
.AC Analysis Statement 157
Option Statement Associated with AC Analysis 158
Statement Defining a Small-Signal AC Source 159
Synopsis of Small-Signal AC Analysis 160
Amplitude of Small-Signal AC Sources 161
Phase Delay of Small-Signal AC Sources 162
Sample Waveforms of AC Sources
Continuous Domain 162
Sample Waveforms of AC Sources
Discrete Domain 163

Continuous and Discrete Domain Differences..... 165
AC Analysis Behaviour of Time-Varying Sources .. 165
Behaviour of AC Sources in Transient and POP.... 166
Example of Applying the AC Analysis Tool 166

Chapter 1 Introduction

Overview

This manual provides detailed reference material for the SIMPLIS (SIMulation for Piecewise-LInear System) simulation package. It is intended for those who want to develop a more in-depth understanding of this software package.

SIMPLIS is a computer software package specifically designed for the simulation and analysis of switching power supplies. In a typical switching power system, the transistors and the diodes function as switches, allowing the system to be characterized by a cyclical sequence of linear circuit topologies. By taking advantage of the repetitive piecewise-linear structure of such systems, SIMPLIS is able to perform the simulation in an efficient and accurate manner.

In order to avoid potential problems caused by simple syntax errors, SIMPLIS automatically checks the syntax of the input file provided by the user. If syntax errors are detected in the input file, error messages are recorded in a file for the user to inspect and correct the errors. This feature allows the user to detect and correct errors quickly and efficiently.

SIMPLIS-TX is a two-pass time-domain simulator. In the first pass of a simulation run, only the data pertaining to the state of the simulation at the switching instances are saved. In the second pass, called the Post-Simulation Processing run, detailed waveform information is reconstructed from the data generated in the first pass. At the user-interface level, these two separate operations are not distinguishable since they are executed automatically by SIMPLIS. Thus SIMPLIS appears as a one-pass simulator to the user. This internal two-step simulation technique optimizes the simulation speed and versatility of SIMPLIS.

SIMPLIS-POP (Periodic Operating Point) is an analysis tool that accelerates the convergence to the steady-state solution of switch-mode power systems. By taking into consideration the variation of the timing of the intracycle intervals of a switch-mode system with respect to changes in the state vector, the POP analysis tool accurately calculates the steady-state solution.

SIMPLIS-FX is a special small-signal frequency-domain analyzer developed for the analysis of switching power supplies. By calculating the circuit's response, over a range of frequencies, to a small perturbation in the time domain and then using fast fourier analysis techniques, the necessity of developing state-space averaged equivalent circuits is avoided. SIMPLIS-FX accurately computes the frequency response from the same schematic used for time-domain simulation.

Organization of this User Manual

This user manual is made up of the following chapters.

Chapter 1 is the introduction (this chapter).

Chapters 2-6 explain the syntax and format of input files:
Chapter 2 describes the organization and the basic rules governing the input file.

Chapters 3 and 4 cover the format rules for specifying various device types.
Chapter 5 explains the definition of subcircuits.
Chapter 6 describes control statements.

Chapters 7-9 explains the execution of SIMPLIS.
Chapter 7 explains the SIMPLIS commands and command-line options.
Chapter 8 gives a synopsis of the data files generated by SIMPLIS.
Chapter 9 provides a set of examples illustrating the capabilities, features and the simulation options.

Chapter 10 explains the SIMPLIS-POP (Periodic Operating Point) analysis tool.
Chapter 11 explains the SIMPLIS-FX small-signal frequency-domain analyzer.

Chapter 2 Input File Organization

Overview

SIMPLIS uses a single text input file to define:

1. The interconnections and components forming the circuit,
2. The options that apply in the analysis, and
3. The specific analyses to be performed.

In this chapter, "General Rules for the Input File" on page 10 defines the general rules for the input file and "Organization of the Input File" on page 17 explains the organization of the input file.

General Rules for the Input File

Statements and Continuation of Statements

The input file for the SIMPLIS package is organized into different statements. Ordinarily, the end of a line signifies the end of a statement. However, a statement can be continued on to the next line by using the line continuation character, the plus sign (`+'). Refer to "Continuation of Statements" on page 11 for more about the continuation of statements.

Blank Characters

Within a single statement, the data are organized into different fields. Fields are separated by one or more spaces or tabs. Spaces and tabs are called blank characters. While blank characters must be used to separate different fields, the presence of at least one blank character between two groups of non blank characters does not always mean the two groups of characters belong to two different fields. Blank characters may be present within a single field. In general, blank characters are used to separate different fields or to improve the readability of the input file.

Blank Lines

Blank lines can be placed anywhere in the input file to make the file more readable. Blank lines have no effect on the execution of the program.

Comment Statements

Comment statements are used to make the input file more readable. A comment is identified by an asterisk (`*') as the first character, at the beginning of a line. Comment statements which are several lines long must have a comment character at the beginning of each line. Comment statements are ignored during program execution.

In-line Comments

An in-line comment is a comment that starts in the middle of a line. The in-line comment is identified by the semicolon character, (`;'), placed at the beginning of the comment section. The portion of the line to the right of the in-line comment symbol is ignored by SIMPLIS. For example, in SIMPLIS, the following two lines are equivalent:

```
RC 1 0 1K ;esr of c1

RC 1 0 1K
```

Continuation of Statements

A statement can be continued to the next line by using the line continuation character, the plus sign (`+'). Any line whose first non blank character is the line continuation character is considered the continuation of the previous line. For example, the following lines form one single statement:

```
V1 1 0 PUL V1=0 V2=1
+ FREQ=1MEG DRATIO=0.1 DELAY=0.1
+ OFF_UNTIL_DELAY=YES
```

A comment statement cannot be followed by a line continuation statement.

Device Names

The name of a device is formed by the concatenation of two parts: the element keyword and the individual name. The element keyword is a character string of one or two characters. The individual name is a character string of arbitrary length. You should keep the individual name descriptive and short (no more than sixteen characters).

Element Keyword

The table below shows the relationship between each element keyword and the type of corresponding circuit elements.

Element Keyword	Type of Element
R	Linear Resistor
L	Linear Inductor
C	Linear Capacitor
V	Independent Voltage Source
I	Independent Current Source
M	Linear Mutual Inductance
E	Linear Voltage-Controlled Voltage Source

SIMPLIS Element Types

Element Keyword	Type of Element
G	Linear Voltage-Controlled Current Source
H	Linear Current-Controlled Voltage Source
F	Linear Current-Controlled Current Source
!T	Ideal Transformer
Q	Simple Transistor Switch
S	Simple Switch
!R	Piecewise-Linear Resistor
!L	Piecewise-Linear Inductor
!C	Piecewise-Linear Capacitor
!D	Simple Logic Gates
X	Instantiation of subcircuits

SIMPLIS Element Types

Individual Name

An individual name is a string made up of zero or more characters from the following character set:

Alphabetic characters a-z A-Z

Numeric characters 0-9

Underscore _

Examples of Device Names

The following entries are all legal device names for inductors:

```
L La LA L1 L_MAG
```

Model Names and Subcircuit Names

SIMPLIS supports the concept of device models and subcircuits in the input file. The name of a model or a subcircuit is a string that is made up of the same characters as those outlined in See Individual Name for the individual name of a device. In addition, each of the following conditions must also be satisfied:

1. A model name or a subcircuit name must have at least one character.
2. A model or subcircuit name must contain at least one alphabet character.
3. The first character in a model or subcircuit name must not be the underscore character (`_').

Uppercase vs. Lowercase

SIMPLIS is case sensitive to individual device names . For example, the following device names are two different inductors:

```
La
LA
```

However, SIMPLIS is not case sensitive to element keywords . For example, the following names are the same inductor:

```
La
la
```

because the first character (`l' or `L') is the element keyword for a device name.

The interpretation of the model names and subcircuit names follows the same interpretation as the operating system of the host. If the operating system is case sensitive to file names, then SIMPLIS is case sensitive to model names and subcircuit names. If the operating system is not case sensitive to file names, then SIMPLIS is not case sensitive to model names and subcircuit names. For example, UNIX operating systems are case sensitive, while Windows operating systems are not.

Integer Entries

Some input statements may have fields or parameters which are required to be integers. The legal format for an integer entry is:

```
[-]d[d...]
```

where

[-]	is the optional negative sign associated with a negative integer,
d	is a numeral in the range of 0 through 9, and
[d...]	is an optional string of extra digits.

The following are legal entries:

```
-34 0 25 -27 301
```

The following are illegal integer entries:

```
- +1 23. 24.5
```

They are illegal integer entries because

1. The negative sign is not followed by a numeral;
2. +1 begins with a positive sign, which is illegal;
3. The numbers 23. and 24.5 have decimal points.

Floating-point Entries

From time to time, a certain field or parameter in a statement calls for a floating-point entry. A floating-point entry can be typed in several possible formats:

1. Integer format
2. Simple floating-point format
3. Exponential format
4. Engineering format

Integer Format in Floating-point Entries

Whenever a floating-point entry is expected, the entry can be typed in the integer format as defined in “Integer Entries” on page 13 if the corresponding entry turns out to be an integer. For example, if 34 is to be typed as a floating-point entry, it can be typed as 34 without the accompanying decimal point.

Simple Floating-point Format

The simple floating-point format is defined as

[-]d[d...].[d...] or [-].d[d...]

where

[-]	is the negative sign associated with a negative integer,
d	is a numeral in the range of 0 through 9,
.	is the decimal point, and
[d...]	is an optional string of extra digits.

Examples of the use of the simple floating-point format are:

```
9.37 -0.5 1001.76
```

Similar to an integer entry, a floating-point entry cannot begin with a positive sign.

Exponential Floating-point Format

The exponential floating-point format is defined as

[if]E[+-]d[d] or [sfpf]E[+-]d[d]

where

[if]	is a number in the integer format,
[sfpf]	is a number in the simple floating-point format,
E	is either the character e or the character E,
[+ -]	is either the positive sign or the negative sign,

d	is a numeral in the range of 0 through 9, and
[d]	is an optional extra digit in the exponent.

The following examples are equivalent entries:

```
27000 2.7e+04 27E+3
```

Engineering Floating-point Format

The engineering floating-point format is defined as

[if]S or [sfpf]S or [efpf]S

where

[if]	is a number in the integer format,
[sfpf]	is a number in the simple floating-point format,
[efpf]	is a number in the exponential floating-point format, and
S	is one of the character strings used to represent one of the scale factors. The string can be entered in either lower or upper case.

The following examples are equivalent entries:

```
27k 27K 27000 27000 2.7E+4 27e-03MEG
```

The table below shows all the engineering prefixes recognized by SIMPLIS, and their corresponding scale values.

Symbol	Prefix	Scale Factor
F	femto	10 -15
P	pico	10 -12
N	nano	10 -9
U	micro	10 -6
M	milli	10 -3
K	kilo	10 +3
MEG	mega	10 +6
G	giga	10 +9
T	tera	10 +12

Prefix Types

Illustrations of Legal and Illegal Floating-point Entries

The following entries are all valid floating-point entries:

```
0 -3 3. 0.3 31.12 -.12E-06 3.12e+3 1.1K -150U
```

The entries 0 and -3 are in integer format. The entries 3., 0.3, and 31.12 are in simple floating-point format. The entries -.12E-06 and 3.12e+3 are in exponential format. The entries 1.1K and -150U are in engineering format.

The following are illegal floating-point entries:

```
+0.7 3.12E+345 4.7 K
```

They are illegal because

1. +0.7 contains the illegal positive sign;
2. The exponent in 3.12E+345 is more than two digits long;
3. 4.7 K has an extra space between the number 4.7 and the scale factor K.

Units

SIMPLIS works with System Internationale (SI) units. The table below gives a summary of all units expected for different types of variables. Units are not allowed to be specified with the values of the corresponding variables. For example, a capacitance of 1.25 microfarads may be represented by 1.25U or 0.00000125, but not 1.25UF or 0.00000125F.

Variable	Units
Time	second
Resistance	ohm
Capacitance	farad
Inductance	henry
Voltage	volt
Current	ampere
Charge	coulomb

Unit Types

Length of Fields and Lines

Each field of entry should be restricted to no more than 80 characters long. Each input line should be restricted to no more than 160 characters long. This restriction does not limit the length of a statement since it can continue over several lines through the line continuation character.

Organization of the Input File

SIMPLIS supports the concept of a main circuit and subcircuits in the definition of the system to be analyzed. A subcircuit can be nested within another subcircuit for up to 20 levels of nesting, with the main circuit considered as the first level of nesting. Since the definitions for the main circuit and a subcircuit are similar, the term general circuit is used here to represent either the main circuit or a subcircuit.

General Circuit

The general circuit is defined by the following statements:

1. Start Circuit Statement
2. Comment Statements
3. Device Statements
4. Model Statements
5. Subcircuit Definition Statements
6. Control Statements
7. End Circuit Statement

Subcircuit Definition Statements defining a subcircuit follow the same pattern of statements outlined here for the general circuit. The forms of the Start Circuit Statement and End Circuit Statement depend on whether the general circuit is the main circuit or a subcircuit. The forms of the rest of the statements remain the same for both the main circuit and subcircuits.

Sequence of Statements

The scope of definition for a general circuit begins at the Start Circuit Statement and stops at the End Circuit Statement, inclusively. Statements within the scope of definition of a general circuit can be placed in any sequence without any effect on the reading of the input file, with the following exceptions:

1. In the definition of a general circuit, the Start Circuit Statement and the End Circuit Statement must be the first and the last statements, respectively.
2. Analysis statements are special control statements. The order in which analysis statements appear in the input file determines the order in which SIMPLIS performs different analyses.

Main Circuit

Title Statement (Start of Main Circuit Statement)

The first line in the input file is the Title Statement. This statement is the Start Circuit Statement for the main circuit, and it is copied to some of the data files generated by SIMPLIS for annotation purpose. The Title Statement must be only one line long. It cannot be extended over additional lines by using the line continuation character.

.END Statement (End of Main Circuit Statement)

The first statement in the input file that has the first field matching the keyword .END is the end of main circuit statement. This statement is the End Circuit Statement for the main circuit. Any input lines following this .END statement in the input file are ignored by SIMPLIS. Since .END is a keyword, its interpretation is case insensitive. No other fields are allowed in this statement. It cannot be extended over additional lines by using the line continuation character

Subcircuit

The details of the SIMPLIS subcircuit feature are explained in “Subcircuit Definition” on page 86. The following two subsections give a brief outline of how the subcircuits are defined.

.SUBCKT Statement (Start of Subcircuit Statement)

Any statement whose first field matches the keyword .SUBCKT starts the definition of a subcircuit. The .SUBCKT statement is the Start Circuit Statement for a subcircuit. The keyword .SUBCKT is followed by the name of the subcircuit and a group of node names.

.ENDS Statement (End of Subcircuit Statement)

A statement whose first field matches the keyword .ENDS is the end of the subcircuit statement. This statement is the End Circuit Statement for a subcircuit. The .ENDS statement can have two forms. In the first form, the .ENDS keyword is the only field in the statement. In the second form, the .ENDS keyword is followed by the name of a proper subcircuit.

General Statements

Comment Statements

See “Comment Statements” on page 10 and “In-line Comments” on page 11 for an explanation on the use of comments in the input file.

Device Statements

A device statement defines the parameter values of the device and indicates how it is connected to the circuit. When a model name or initial condition is required for a device, they are also defined in the device statement. “Device Statements” on page 20 provides a detailed description of the syntax of device statements.

Model Statements

The model statement defines the parameters associated with a particular device model. Once a model is defined, it allows SIMPLIS to insert the model characteristics for every device associated with that model name. The Model Statement always starts with the keyword .MODEL as the first field in the statement. The following is a typical model statement for a diode, modeled as a piecewise-linear resistor:

```
. MODEL MD1M VPWLR NSEG=2 X0=0 Y0=0 X1=0.7 Y1=10U
```

```
+ X2=0.8 Y2=1
```

“Model Statements” on page 50 describes the syntax for model statements.

Control Statements

Control Statements start with the period character (`.'), and can be classified into one of the following types:

1. Options
2. Initial conditions
3. Resource limits
4. Analyses

Although all control statements start with a period character, not all statements which start with a period are control statements. For example, statements beginning with keywords such as .MODEL, .SUBCKT, .END and .ENDS which start with a period character are not control statements. “Control Statements” on page 95 explains the meaning and syntax of all control statements supported by SIMPLIS.

Chapter 3 Device Statements

Overview

Device Statement Format

The device statement defines how a device is connected in the circuit, and lists the values for the individual device parameters. If a device requires a device model or some initial condition, such information is also defined in the device statement. The format for a device statement is defined as follows:

DeviceName NodeName {*Values*|*ModelName*} [*InitConds*]

where

DeviceName	is a legal device name
NodeName	is a sequence of legal node names
Values	is either a floating-point entry to stand for value or a sequence of parameter assignments
ModelName	is the legal name of a compatible model. The symbol {*Values*|*ModelName*} means Values and ModelName are mutually exclusive. If a device requires a model name, no Values would be given in the device statement and vice versa
InitConds	is a sequence of legal initial condition specifications. The symbol [InitConds] means the presence of the InitConds fields is optional since only some devices require initial conditions.

The individual fields in each device statement must appear exactly in the order indicated in this chapter. Any different sequence will cause SIMPLIS to misinterpret the statement or generate an error message.

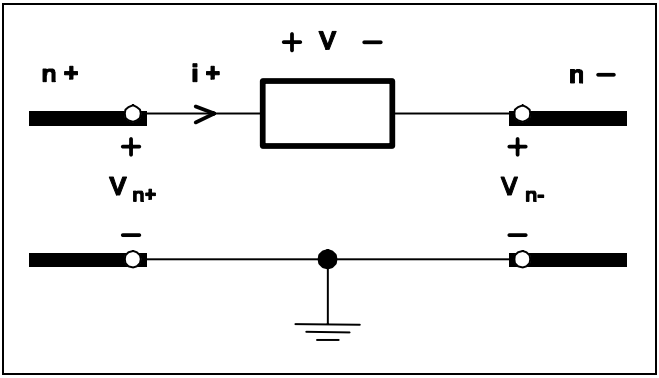
Node Names

Each node in the circuit must be assigned a unique name. A legal node name is either a positive integer or zero. Remember that node 0 is traditionally reserved to represent the ground node. Also note that:

1. SIMPLIS does not require the presence of node 0 in the system unless the user wants to inspect the voltage of a particular node with respect to a certain ground node
2. SIMPLIS does not require every node in the system to be connected to each other, allowing the system to have isolated subsystem. However, error messages will be generated if the user instructs SIMPLIS to determine the voltage between two electrically isolated nodes.

Voltage and Current Polarity Conventions

Most of the circuit elements discussed in this chapter are two-terminal elements. For any general two-terminal element, there is a positive node n+ and a negative node n- as shown in the diagram below:



3.1 Definition of the voltage and the current association for a two-terminal element

Whenever the voltage across a two-terminal element is mentioned in this manual, it refers to the voltage measured at the positive node with respect to the voltage at the negative node. For example, the voltage of the generalized two-terminal device in See Definition of the voltage and the current associated for a two-terminal element is equal to

V = Vn+ - Vn-

as measured from the n+ terminal to the n- terminal.

When the current through a two-terminal device is mentioned in this manual, it refers to the current measured in the direction from the positive node through the element to the negative node. Thus, a positive current indicates that there is a net flow of charge into the positive node, through the two-terminal element, and then out of the negative node.

With these sign conventions for voltages and currents, a positive voltage-current product indicates that electric power is instantaneously flowing into the corresponding two-terminal element, whether it is a voltage or current source, a resistor, or any other two-terminal element.

Parameter Assignments

Parameters such as the initial conditions for devices are entered in a format called parameter assignments. The format is

KEYWORD=value

where

KEYWORD	is the corresponding keyword representing the descriptive name of the parameter,
=	is the equal sign (`='), and
value	is the value assigned to the parameter.

To make the input file more readable, blank characters can appear before and/or after the equal sign (`='). In addition, if the equal sign is the last significant character in the current line, the value can appear in the next line through the use of the line continuation character, the plus sign (`+').

Controlling Devices

The four controlled sources, the simple transistor switch, and the simple switch are each controlled by a controlling variable that is external to the device. There are three types of controlling variables:

1. A differential voltage across two nodes in the same circuit,
2. A branch voltage across the positive and negative nodes of a controlling device in the same circuit, or
3. A current through a controlling device in the same circuit.

A controlling device can be any one of the following device types:

Linear resistors
Linear inductors
Linear capacitors
All types of independent voltage sources
All types of independent current sources
All four types of linear controlled sources
Simple transistor switches
Simple switches
Piecewise-linear resistors
Piecewise-linear inductors
Piecewise-linear capacitors

SIMPLIS Device Types

Linear Resistors

The format for a linear resistor is:

R*name* *n+* *n-* *value*

where

R	is the one-character element keyword "R" for linear resistors
name	is the individual name of the device

n+	is the name of the positive node, and is a nonnegative integer
n-	is the name of the negative node, and is a nonnegative integer
value	is a floating-point number assigned as the value of the resistance (in ohms). This value can be positive, zero, or negative

Linear Inductors and Capacitors

The formats for a linear inductor and a linear capacitor are:

L*name* *n+* *n-* *value* IC=*init_cond*

C*name* *n+* *n-* *value* IC=*init_cond*

where

L	is the one-character element keyword "L" for linear inductors
C	is the one-character element keyword "C" for linear capacitor
name	is the individual name of the device
n+	is the name of the positive node, and is a nonnegative integer
n-	is the name of the negative node, and is a nonnegative integer
value	is a floating-point number assigned as the value of the inductance (in henries) for an inductor or the value of the capacitance (in farads) for a capacitor. The value can be positive or negative, but not zero
IC=	is the three-character keyword "IC="
init_cond	is a floating-point number assigned as the value of initial condition. It is the initial current (in amperes) for an inductor or the initial voltage (in volts) for a capacitor

Independent Voltage and Current Sources

DC Sources

The formats for the dc sources are:

V*name* *n+* *n-* DC *value*

I*name* *n+* *n-* DC *value*

where

V	is the one-character element keyword "V" for independent voltage sources
I	is the one-character element keyword "I" for independent current sources
name	is the individual name of the device

n+	is the name of the positive node, and is a nonnegative integer
n-	is the name of the negative node, and is a nonnegative integer
DC	is the two-character keyword "DC" to signify that this is a DC source
value	is a floating-point number assigned as the source value. It is the voltage across the source element (in volts) for a dc voltage source or the current through the source (in amperes) for a dc current source

Sawtooth Sources

The format for the independent sawtooth voltage source is:

V*name* *n+* *n-* SAW V1=*v1* V2=*v2*
+ FREQ=*freq* DELAY=*delay*
+ OFF_UNTIL_DELAY={YES|NO}

The format for the independent sawtooth current source is:

I*name* *n+* *n-* SAW V1=*v1* V2=*v2*
+ FREQ=*freq* DELAY=*delay*
+ OFF_UNTIL_DELAY={YES|NO}

where

V	is the one-character element keyword "V" for independent voltage sources
I	is the one-character element keyword "I" for independent current sources
name	is the individual name of the device
n+	is the name of the positive node, and is a nonnegative integer
n-	is the name of the negative node, and is a nonnegative integer
SAW	is the three-character keyword "SAW" to signify that this is a sawtooth source
V1=	is the three-character keyword "V1=" representing the source value at the start of a normal cycle
v1	is a floating-point number assigned as the value of V1 (in volts) for a voltage source and the value of V1 (in amperes) for a current source
V2=	is the three-character keyword "V2=" representing the source value at the end of a normal cycle
v2	is a floating-point number assigned as the value of V2 (in volts) for a voltage source and the value of V2 (in amperes) for a current source
FREQ=	s the five-character keyword "FREQ="

freq	s a positive floating-point number assigned as the frequency of this source (in hertz)
DELAY=	s the six-character keyword "DELAY="
delay	s a floating-point number assigned as the value of DELAY (in seconds)
OFF_UNTIL_DELAY=	is the sixteen-character keyword "OFF_UNTIL_DELAY="
YES	s the three-character keyword "YES"
NO	s the two-character keyword "NO"

The plus characters shown in the format definition are not necessary if carriage returns are not used in the statement. The plus characters and the carriage returns have been added to break the statements over different lines to make them easier to read.
Using the function s(t) to represent the voltage across the voltage source or the current through the current source, the value of the source function s(t) in the diagram below for $t > delay$ is defined as:

$$s(t)=v1+[(v2-v1)(t-delay)]/T$$

for $delay < t < (delay + T)$

and:

$$s(t)=s(t-T)$$

for $(delay + T) < t$

where

T	=1/(freq) is the period of the waveform

The source function s(t) for t < delay is defined as follows

$$s(t)=s(t+T)$$

for 0 < t < delay and OFF_UNTIL_DELAY=NO

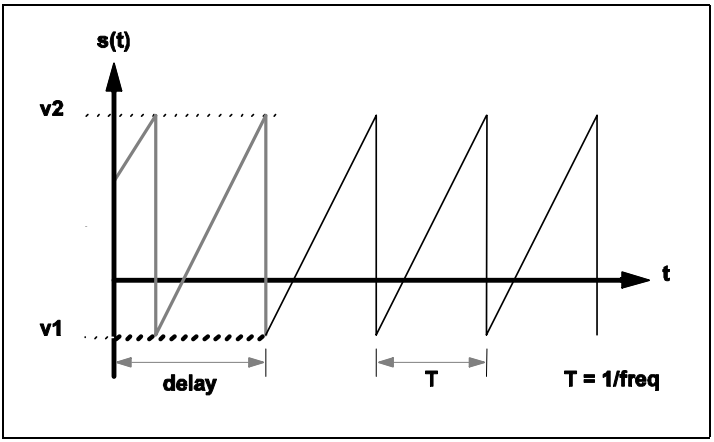
and

$$s(t)=v1$$

for 0 < t < delay and OFF_UNTIL_DELAY=YES

Whether the delay is positive or negative, the time-domain transient analysis performed by SIMPLIS always starts with the time variable set equal to 0.0.

The diagram below shows the waveforms of a sawtooth source. For t < delay and OFF_UNTIL_DELAY=YES, the waveform s(t) is shown in bold dashed line. For t < delay and OFF_UNTIL_DELAY=NO, the waveform s(t) is shown in heavy gray line.



3.2 Waveform s(t) of a sawtooth source

Triangular Sources

The formats for independent triangular voltage and current sources are:

V*name* *n*+ *n*- TRI V1=*v1* V2=*v2*
+ FREQ=*freq* DRATIO=*dratio* DELAY=*delay*
+ OFF_UNTIL_DELAY={YES|NO}

and

I*name* *n*+ *n*- TRI V1=*v1* V2=*v2*
+ FREQ=*freq* DRATIO=*dratio* DELAY=*delay*
+ OFF_UNTIL_DELAY={YES|NO}

where

V	is the one-character element keyword "V" for independent voltage sources
I	is the one-character element keyword "I" for independent current sources
name	is the individual name of the device
n+	is the name of the positive node, and is a nonnegative integer
n-	is the name of the negative node, and is a nonnegative integer
TRI	is the three-character keyword "TRI" to signify that this is a triangular source
V1=	is the three-character keyword "V1=" representing the source at the start of a normal cycle

v1	is a floating-point number assigned as the value of V1 (in volts) for a voltage source or the value of V1 (in amperes) for a current source
V2=	is the three-character keyword "V2=" representing the source at the end of a normal cycle
v2	is a floating-point number assigned as the value of V2 (in volts) for a voltage source or the value of V2 (in amperes) for a current source
FREQ=	is the five-character keyword "FREQ="
freq	is a positive floating-point number assigned as the frequency of this source (in hertz)
DRATIO=	is the seven-character keyword "DRATIO="
dratio	is a dimensionless floating-point number between 0.0 and 1.0, exclusively, assigned as the value of DRATIO
DELAY=	is the six-character keyword "DELAY="
delay	is a floating-point number assigned as the value of DELAY (in seconds)
OFF_UNTIL_DELAY=	is the sixteen-character keyword "OFF_UNTIL_DELAY="
YES	is the three-character keyword "YES"
NO	is the two-character keyword "NO"

The source function s(t) for $t > delay$ is defined as follows:

$$s(t)=v1+[(v2-v1)(t-delay)]/t1$$

for delay < t < (delay + t1)

$$s(t)=v2+[(v1-v2)(t-delay-t1)]/(T-t1)$$

for $(delay + t1) < t < (delay + T)$

and

$$s(t)=s(t-T)$$

for $(delay + T) < t$

where

T	=1/(freq) is defined as the period of the waveform
t1	=DRATIO*T is the duration in a period of the waveform where the source value is moving from the value of *v1* to the value of *v2*.

The source function s(t) for t < delay is defined as follows:

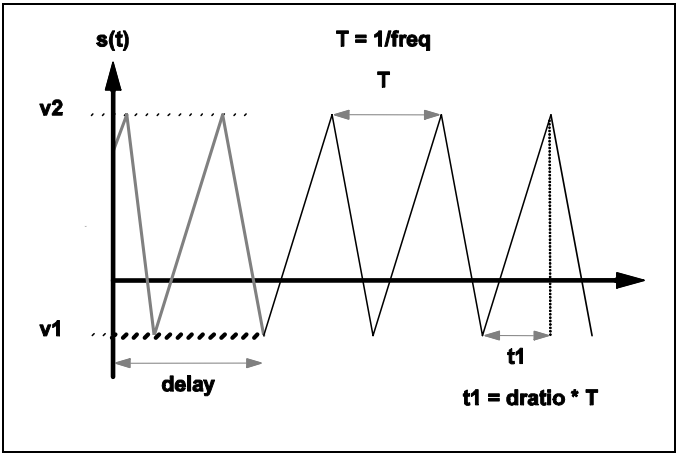
s(t)=s(t+T)

for 0 < t < delay and OFF_UNTIL_DELAY=NO

and

s(t)=v1 for 0 < t < delay and OFF_UNTIL_DELAY=YES.

Again, whether the delay is positive or negative, the time-domain transient analysis performed by the simulation always starts with the time variable set equal to 0.0. The diagram below shows the waveform, s(t), of a typical triangular source. For t < delay and OFF_UNTIL_DELAY=YES, the waveform s(t) is shown in bold dashed line. For t < delay and OFF_UNTIL_DELAY=NO, the waveform s(t) is shown in heavy gray line.



3.3 Waveform s(t) of a triangular source

Square Wave Sources

The formats for independent square-wave voltage and current sources are:

V*name* *n+* *n-* SQU V1=*v1* V2=*v2* FREQ=*freq*
+ DELAY=*delay* OFF_UNTIL_DELAY={YES|NO}

and

I*name* *n+* *n-* SQU V1=*v1* V2=*v2* FREQ=*freq*
+ DELAY=*delay* OFF_UNTIL_DELAY={YES|NO}

where

V is the one-character element keyword "V" for independent voltage sources

I	is the one-character element keyword "I" for independent current sources
name	is the individual name of the device
n+	is the name of the positive node, and is a nonnegative integer
n-	is the name of the negative node, and is a nonnegative integer
SQU	is the three-character keyword "SQU" to signify that this is a square source
V1=	is the three-character keyword "V1=" representing the source at the start of a normal cycle
v1	is a floating-point number assigned as the value of V1 (in volts) for a voltage source and the value of V1 (in amperes) for a current source
V2=	is the three-character keyword "V2=" representing the source at the end of a normal cycle
v2	is a floating-point number assigned as the value of V2 (in volts) for a voltage source and the value of V2 (in amperes) for a current source
FREQ=	is the five-character keyword "FREQ="
freq	is a positive floating-point number assigned as the frequency of this source (in hertz)
DELAY=	is the six-character keyword "DELAY="
delay	is a floating-point number expressing DELAY (in seconds)
OFF_UNTIL_DELAY=	is the sixteen-character keyword "OFF_UNTIL_DELAY="
YES	is the three-character keyword "YES"

NO is the two-character keyword "NO"

The source function s(*t*) for t > delay is defined as follows:

s(*t*)=*v2* for *delay* < *t* < (*delay* + T/2):

s(*t*)=*v1*for (*delay* + T/2) < *t* < (*delay* + T):

s(*t*)=s(*t*-T)for (*delay* + T) < *t*:

The source function s(*t*) for t < *delay* is defined as follows

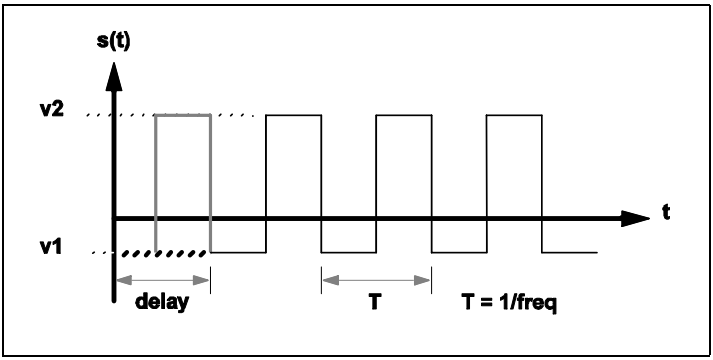
s(*t*)=s(*t*+T) for 0 < *t* < *delay* and OFF_UNTIL_DELAY=NO

s(*t*)=v1for 0 <*t* < *delay* and OFF_UNTIL_DELAY=YES

where

T =1/*freq* T is defined as the period of the waveform

The waveform s(*t*) of a typical squarewave source is illustrated in the diagram below. For *t* < *delay* and OFF_UNTIL_DELAY=YES, the waveform s(*t*) is shown in bold dashed line. For *t* < *delay* and OFF_UNTIL_DELAY=NO, the waveform s(*t*) is shown in heavy gray line.



3.4 Waveform s(t) of a squarewave source

Pulse Sources with Zero Rise and Fall Times

The formats for the independent rectangular pulse sources are

V*name* *n*+ *n*- PUL V1=*v1* V2=*v2* FREQ=*freq*
+ DRATIO=*dratio* DELAY=*delay*
+ OFF_UNTIL_DELAY={YES|NO}

and

I*name* *n*+ *n*- PUL V1=*v1* V2=*v2* F REQ=*freq*
+ DRATIO=*dratio* DELAY=*delay*
+ OFF_UNTIL_DELAY={YES|NO}

where

V	is the one-character element keyword "V" for independent voltage sources
I	is the one-character element keyword "I" for independent current sources
name	is the individual name of the device
n+	is the name of the positive node, and is a nonnegative integer
n-	is the name of the negative node, and is a nonnegative integer

PUL	is the three-character keyword "PUL" to signify that this is a rectangular pulse source
V1=	is the three-character keyword "V1=" representing the source at the start of a normal cycle
v1	is a floating-point number assigned as the value of V1 (in volts) for a voltage source and the value of V1 (in amperes) for a current source
V2=	is the three-character keyword "V2=" representing the source at the end of a normal cycle
v2	is a floating-point number assigned as the value of V2 (in volts) for a voltage source and the value of V2 (in amperes) for a current source
FREQ=	is the five-character keyword "FREQ="
freq	is a positive floating-point number assigned as the frequency of this source (in hertz)
DRATIO=	is the seven-character keyword "DRATIO="
dratio	is a dimensionless floating-point number between 0.0 and 1.0, exclusively, assigned as the value of DRATIO
DELAY=	is the six-character keyword "DELAY="
delay	is a floating-point number assigned as the value of DELAY (in seconds)
OFF_UNTIL_DELAY=	is the sixteen-character keyword "OFF_UNTIL_DELAY="
YES	is the three-character keyword "YES"
NO	is the two-character keyword "NO"

The source function s(t) for t > delay is defined as follows

$s(t)=v2$ for $delay < t < (delay + t1)$

$s(t)=v1$ for $(delay + t1) < t < (delay + T)$

$s(t)=s(t-T)$ for $(delay + T) < t$

where

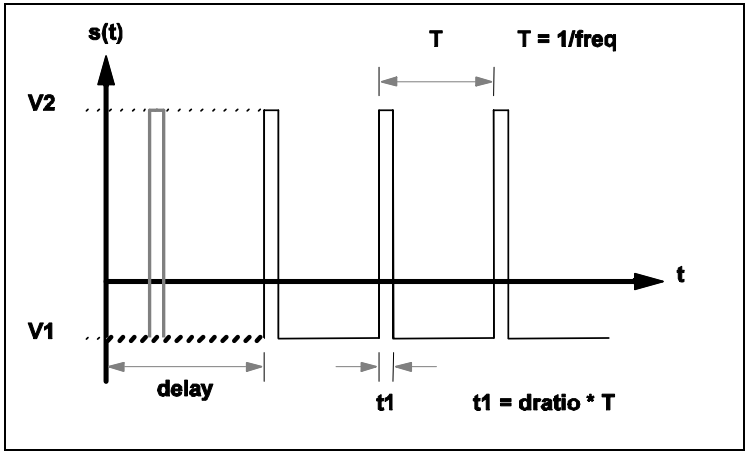
T	=1/freq T is defined as the period of the waveform
t1	=DRATIO*T t1 is the duration in a period of the waveform where the source value is equal to *v2*

The source function s(t) for t < delay is defined as follows:

$s(t)=s(t+T)$ for $0 < t < delay$ and OFF_UNTIL_DELAY=NO

s(*t*)=*v1* for 0 < *t* < *delay* and OFF_UNTIL_DELAY=YES

The waveform s(*t*) of a typical rectangular pulse source is shown in the diagram below. For t < delay and OFF_UNTIL_DELAY=YES, the waveform s(t) is shown in bold dashed line. For t < delay and OFF_UNTIL_DELAY=NO, the waveform s(t) is shown in heavy gray line.



3.5 Waveform s(t) of a pulse-wave source.

Sinusoidal Sources

The formats for defining independent sinusoidal voltage and current sources are:

V*name* *n*+ *n*- SIN VOFFSET=*voff* APEAK=*apeak*
+ FREQ=*freq* {TDELAY=*tdelay*|PDELAY=*pdelay*}
+ OFF_UNTIL_DELAY={YES|NO} DAMP_COEF=*damp_coef*

and

I*name* *n*+ *n*- SIN VOFFSET=*voff* APEAK=*apeak*
+ FREQ=*freq* {TDELAY=*tdelay* |PDELAY=*pdelay*}
+ OFF_UNTIL_DELAY={YES|NO} DAMP_COEF=*damp_coef*

where

V	is the one-character element keyword "V" for independent voltage sources
I	is the one-character element keyword "I" for independent current sources
name	is the individual name of the device
n+	is the name of the positive node, and is a nonnegative integer

n-	is the name of the negative node, and is a nonnegative integer
SIN	is the three-character keyword "SIN" to signify that this is a sinusoidal source with possible damping
VOFFSET=	is the eight-character keyword "VOFFSET=" representing the DC offset of the source
voff	is a floating-point number assigned as the DC offset value (in volts) for a voltage source and the DC offset value (in amperes) for a current source
APEAK=	is the six-character keyword "APEAK=" representing the amplitude of the source at t = tdelay
apeak	is a nonnegative floating-point number assigned as volts for a voltage source and amperes for a current source
FREQ=	is the five-character keyword "FREQ=" representing the frequency of the source
freq	is a positive floating-point number assigned as the frequency of this source (in hertz)
TDELAY=	is the seven-character keyword "TDELAY=" representing the time delay of the source
tdelay	is a floating-point number assigned as the time delay (in seconds)
PDELAY=	is the seven-character keyword "PDELAY=" representing the phase delay of the source
pdelay	is a floating-point number assigned as the phase *delay* (in degrees). The specification of TDELAY and PDELAY are mutually exclusive
OFF_UNTIL_DELAY=	is the sixteen-character keyword "OFF_UNTIL_DELAY="
YES	is the three-character keyword "YES"
NO	is the two-character keyword "NO"
DAMP_COEF=	is the ten-character keyword "DAMP_COEF=" representing the damping coefficient of the source
damp_coef	is a floating-point number assigned as the damping coefficient (in 1/seconds)

The source function s(t) for all t is

$$s(t)=voff+apeak.e^{-damp_coef.(t-tdelay)}.\sin(2.\pi.freq.(t-tdelay))$$

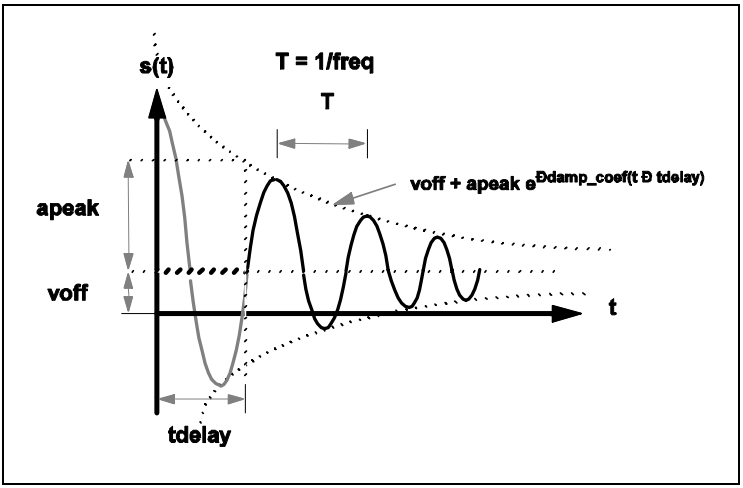
The value of *tdelay*, computed from the value of *pdelay* (if *pdelay* is given), is *tdelay*=(*pdelay*)/(360**freq*).

If OFF_UNTIL_DELAY is assigned a value of YES, then the value of s(*t*) for *t* < *tdelay* is modified to:

s(*t*)=*voff* for *t* < *tdelay*

The waveform s(*t*) of a typical sinusoidal source is shown in the diagram below.

For *t* < *delay* and OFF_UNTIL_DELAY=YES, the waveform s(*t*) is shown in bold dashed line. For *t* < *delay* and OFF_UNTIL_DELAY=NO, the waveform s(*t*) is shown in heavy grey line.



3.6 Waveform s(t) of a sinusoidal source

Cosinusoidal Source

The formats for independent cosinusoidal voltage and current sources are:

V*name n+ n-* COS VOFFSET=*voff* APEAK=*apeak*
+ FREQ=*freq* {TDELAY=*tdelay*|PDELAY=*pdelay*}
+ OFF_UNTIL_DELAY={YES|NO}
+ DAMP_COEF=*damp_coef*

and

I*name n+ n-* COS VOFFSET=*voff* APEAK=*apeak*
+ FREQ=*freq* {TDELAY=*tdelay*|PDELAY=*pdelay*}
+ OFF_UNTIL_DELAY={YES|NO}
+ DAMP_COEF=*damp_coef*

where

V	is the one-character element keyword "V" for independent voltage sources
I	is the one-character element keyword "I" for independent current sources

name	is the individual name of the device
n+	is the name of the positive node, and is a nonnegative integer
n-	is the name of the negative node, and is a nonnegative integer
COS	is the three-character keyword "COS" to signify that this is a cosinusoidal source with possible damping
VOFFSET=	is the eight-character keyword "VOFFSET=" representing the DC offset of the source
voff	is a floating-point number assigned as the DC offset value (in volts) for a voltage source and the DC offset value (in amperes) for a current source
APEAK=	is the six-character keyword "APEAK=" representing the amplitude of the source at t = tdelay
apeak	is a nonnegative floating-point number assigned as the amplitude (in volts) for a voltage source and (in amperes) for a current source
FREQ=	is the five-character keyword "FREQ=" representing the frequency of this source
freq	is a positive floating-point number assigned as the frequency (in hertz)
TDELAY=	is the seven-character keyword "TDELAY=" representing the time delay of the source (use of TDELAY and PDELAY are mutually exclusive)
tdelay	is a floating-point number assigned as the time delay (in seconds)
PDELAY=	is the seven-character keyword "PDELAY=" representing the phase delay of the source (use of TDELAY and PDELAY are mutually exclusive)
pdelay	is a floating-point number assigned as the phase delay (in degrees)
OFF_UNTIL_DELAY=	is the sixteen-character keyword "OFF_UNTIL_DELAY="
YES	s the three-character keyword "YES"
NO	s the two-character keyword "NO"
DAMP_COEF=	s the ten-character keyword "DAMP_COEF=" representing the damping coefficient of the source
damp_coef	is a floating-point number assigned as the damping coefficient (in 1/seconds)

The source function s(*t*) for all *t* is

$$s(t)=voff+apeak.e^{-damp_coef.(t\ -\ tdelay)}.\cos(2.\pi.freq.(t-tdelay)).$$

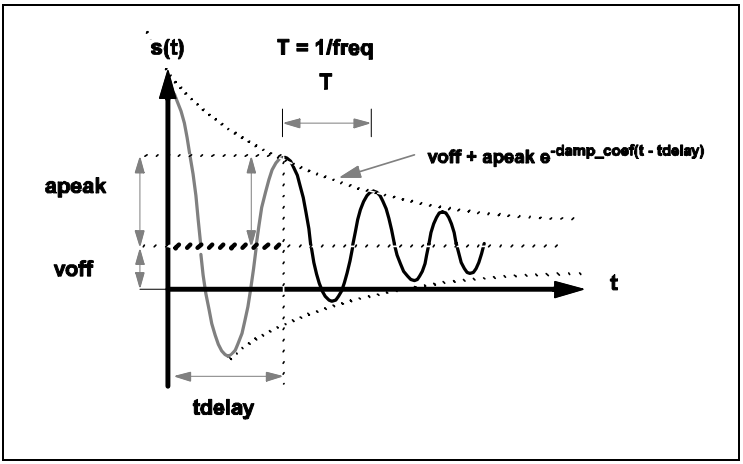
The value of *tdelay*, computed from the value of *pdelay* (if *pdelay* is given), is

tdelay=(*pdelay*)/(360**freq*).

If OFF_UNTIL_DELAY is assigned a value of YES, then the value of s(t) for *t* < *tdelay* is modified to

$$s(t) = voff \qquad \text{for } t < tdelay.$$

The waveform s(*t*) of a typical cosinusoidal source is shown in the diagram below. For t < delay and OFF_UNTIL_DELAY=YES, the waveform s(t) is shown in bold dashed line. For t < delay and OFF_UNTIL_DELAY=NO, the waveform s(t) is shown in heavy grey line.



3.7 Waveform s(t) of a cosinusoidal source

Aperiodic Exponential Pulse Sources

The formats for independent sources with aperiodic exponential pulse waveforms (see waveform diagram below) are as follows:

For a voltage source:

V*name* *n*+ *n*- EXP V1=*v1* V2=*v2*
+ DELAY_R=*delay_r* DELAY_F=*delay_f*
+ TAU_R=*tau_r* TAU_F=*tau_f*

For a current source:

I*name* *n*+ *n*- EXP V1=*v1* V2=*v2*
+ DELAY_R=*delay_r* DELAY_F=*delay_f*
+ TAU_R=*tau_r* TAU_F=*tau_f*

where

V is the one-character element keyword "V" for independent

	voltage sources
I	is the one-character element keyword "I" for independent current sources
name	is the individual name of the device
n+	is the name of the positive node and is a nonnegative integer
n-	is the name of the negative node and is a nonnegative integer
EXP	is the three-character keyword "EXP" to signify that this is an aperiodic single-shot exponential pulse source
V1=	is the three-character keyword "V1=" representing the quiescent value of the source
v1	is a floating-point number assigned as the quiescent value (in volts) for a voltage source and (in amperes) for a current source
V2=	is the three-character keyword "V2=" representing the "pulsed" value of the source
v2	is a floating-point number assigned as the "pulsed" value (in volts) for a voltage source and (in amperes) for a current source
DELAY_R=	is the eight-character keyword "DELAY_R=" representing the time delay of the rising edge of the source, that is, the edge of the waveform when it moves from the quiescent value v1 to the pulsed value v2
delay_r	is a nonnegative floating-point number assigned as the time delay of the rising edge (in seconds)
DELAY_F=	is the eight-character keyword "DELAY_F=" representing the time delay of the falling edge of the source, that is, the edge of the waveform when it moves from v2 to v1
delay_f	is a nonnegative floating-point number assigned as the time delay of the falling edge (in seconds), and must be larger than delay_r
TAU_R=	is the six-character keyword "TAU_R=" representing the time constant of the rising edge of the source
tau_r	is a floating-point number assigned as the value of the time constant of the rising edge (in seconds)
TAU_F=	is the six-character keyword "TAU_F=" representing the time constant of the falling edge of the source
tau_f	is a floating-point number assigned as the value of the time constant of the falling edge (in seconds)

The source function s(*t*) is defined as follows:

$s(t)=v1$ for $t < delay_r$

$s(t)=v2+(v1-v2).e^{-(t - delay_r) / tau_r}$ for $delay_r < t < delay_f$

$$s(t)=v1+(s(delay_f)-v1).e^{-(t\ -\ delay_f)\ /\ tau_f} \quad \text{for } delay_f < t$$

In the case where *tau_r* is equal to 0, the source function rises instantaneously from

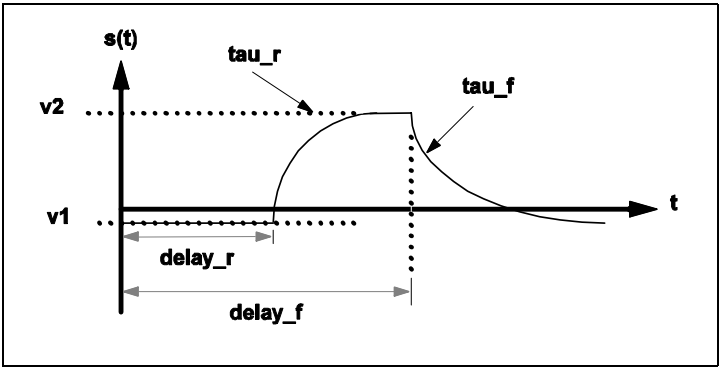
v1 to *v2* at $t = delay_r$

$s(t)=v2$ for $delay_r < t < delay_f$

In the case where *tau_f* is equal to 0, the source function falls instantaneously from

v2 to *v1* at $t = delay_f$
$s(t)=v1$ for $delay_f < t$

The waveform s(*t*) of a typical aperiodic exponential pulse source is shown in the diagram below.



3.8 Waveform s(t) of an exponential pulse source

Aperiodic Piecewise-Linear Sources

An aperiodic piecewise-linear source has its source function s(*t*) defined in terms of a finite number of linear segments. See diagram below for a typical example. The formats for independent aperiodic piecewise linear sources are:

V*name* *n+* *n-* PWL NSEG=*k*
+ X0=*x0* Y0=*y0* ... XK=*xk* YK=*yk*

and

I*name* *n+* *n-* PWL NSEG=*k*
+ X0=*x0* Y0=*y0* ... XK=*xk* YK=*yk*

where

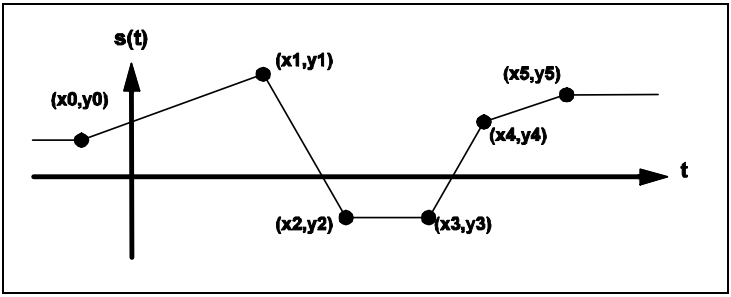
V	is the one-character element keyword "V" for independent voltage sources
I	is the one-character element keyword "I" for independent current sources
name	is the individual name of the device
n+	is the name of the positive node, and is a nonnegative integer
n-	is the name of the negative node, and is a nonnegative integer
PWL	is the three-character keyword "PWL" to signify that this is an aperiodic piecewise-linear source
NSEG=	is the five-character keyword "NSEG=" representing the number of linear segments for this source
k	is an integer defining the number of linear segments and can take on values from 2 to 253, inclusively
X0=	is the keyword "X0=" representing the time (or x axis) coordinate of the start of the first linear segment
x0	is a floating-point number which defines the value of X0 in seconds
Y0=	is the three-character keyword "Y0=" representing the voltage or current (y axis) coordinate of the start of the first linear segment
y0	is a floating-point number which describes the value of Y0 in volts for a voltage source and in amperes for a current source
X1=	is the keyword "X1=" representing the time (or x axis) coordinate of the end of the first linear segment and the start of the second linear segment
x1	is a floating-point number which describes the value of X1 in seconds. x1 is larger than or equal to x0
Y1=	is the keyword "Y1=" representing the voltage or current (y axis) coordinate of the end of the first linear segment and the start of the second linear segment
y1	is a floating-point number which describes the value of Y1 in volts for a voltage source, and in amperes for a current source
X2=	is the keyword "X2=" representing the time (or x axis) coordinate of the end of the second linear segment and the start of the third linear segment
x2	is a floating-point number which defines the value of X2 in seconds. x2 is larger than or equal to x1
Y2=	is the keyword "Y2=" representing the voltage or current (y axis) coordinate of the end of the second linear segment and the start of the third linear segment
y2	is a floating-point number which defines the value of Y2 in volts for a voltage source, and in amperes for a current source, and so

on

The x's and the y's form coordinate pairs in the s(*t*) versus *t* plane. If the source is described by *k* piecewise-linear segments, then (*k*+1) pairs of coordinates are required to define the source, from (*x0*,y0) up to (*xk*,*yk*). The line segment formed by drawing a straight line from (*x0*,*y0*) to (*x1*,*y1*) is the first segment describing the source. The line segment formed by drawing a straight line from (*x1*,*y1*) to (*x2*,*y2*) is the second segment describing the source.

The source value s(t) for t < x0 is set to s(t) = y0. The source value s(t) for t > xk is set to s(t) = yk

An example of the waveform s(t) of a piecewise-linear source is illustrated in the diagram below.



3.9 Example of the waveform s(t) of a typical piecewise-linear source

Mutual Inductances

The format for mutual inductance is:

M-Lname1-Lname2 value

where

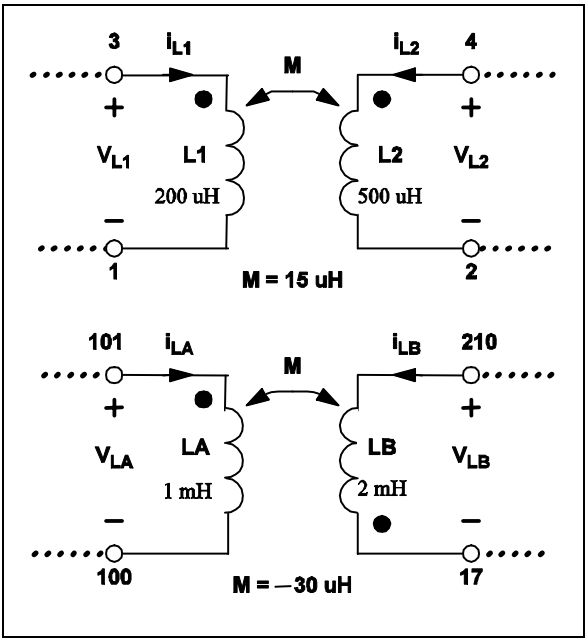
M	is the one-character element keyword "M" for mutual inductors
-	is the hyphen character (`-')
Lname1	is the device name of a linear inductor defined in the current circuit
Lname2	is the device name of another linear inductor defined in the current circuit. Lname1 and Lname2 must refer to different inductors
value	is a floating-point number assigned as the mutual inductance between the two linear inductors

For the schematic shown below, the two mutual inductors are defined in the input file as

```
L1 3 1 200U IC=50M
L2 4 2 500U IC=0
```

```
LA 101 100 1M IC=-30U
LB 210 17 2M IC=10U
M-L1-L2 15U
M-LA-LB -30U
```

The mutual inductance between two linear inductors is positive if the polarity dots appear on the positive nodes of both inductors or if the polarity dots appear on the negative nodes of both inductors. In the example shown below, the mutual inductance between inductors LA and LB is negative because the polarity dot is located at the positive node of LA but the polarity dot is located at the negative node of LB.



3.10 Examples of the definitions for mutual inductances.

Linear Voltage-Controlled Sources

The formats for voltage-controlled sources are:

E*name n+ n- nc+ nc- value*

G*name n+ n- nc+ nc- value*

E*name n+ n- cname value*

G*name n+ n- cname value*

where

E is the one-character element keyword "E" for linear voltage-

	controlled voltage sources
G	is the one-character element keyword "G" for linear voltage-controlled current sources
name	is the individual name of the device
n+	is the name of the positive node of the controlled source, and is a nonnegative integer
n-	is the name of the negative node of the controlled source, and is a nonnegative integer
nc+	is the name of the positive controlling node in the same circuit where the linear voltage-controlled source is being defined
nc-	is the name of the negative controlling node in the same circuit where the linear voltage-controlled source is being defined
cname	is the name of a controlling device in the same circuit where the linear voltage-controlled source is being defined
value	is a floating-point number assigned as the proportionality constant for this controlled source

In the first format, the value of the controlled source is given by

s=*value**v(*nc+*,*nc-*)

where v(*nc+*, *nc-*) represents the voltage of node *nc+* with respect to node *nc-*, and s is the controlled voltage for a controlled voltage source and the controlled current for a controlled current source. In the second format, the value of the controlled source is given by

s=*value**v(*cname*)

where v(*cname*) represents the branch voltage across the positive and negative nodes of the controlling device named "cname".

Linear Current-Controlled Sources

The formats for current-controlled sources are:

H*name n+ n- cname value*

or

F*name n+ n- cname value*

where

H	is the one-character element keyword "H" for linear current-controlled voltage sources
F	is the one-character element keyword "F" for linear current-controlled current sources
name	is the individual name of the device

n+	is the name of the positive node of the controlled source and is a nonnegative integer
n-	is the name of the negative node of the controlled source and is a nonnegative integer
cname	is the name of a controlling device and is not restricted to a voltage source
value	is a floating-point number assigned as the proportionality constant for this controlled source

The value of the controlled source is given by

s=*value**i(*cname*)

where i(*cname*) represents the branch current through the controlling device named "cname".

Ideal Transformers

The format for ideal transformer differs from the typical format defined in “Device Statement Format” on page 20:

!T*name* N_WIND=*k* *n1+* *n1-* N1=*t1* ...
+ *nk+* *nk-* Nk=*tk*

where

!T	is the two-character element keyword "!T" for ideal transformers
name	is the individual name of the device
N_WIND=	is the seven-character keyword "N_WIND=" representing the number of windings in the transformer
k	is a positive integer assigned as the number of windings and can assume any integral value from 2 to 255, inclusively
n1+	is a nonnegative integer to represent the node name of the "dotted" terminal of winding 1
n1-	is a nonnegative integer to represent the node name of the "undotted" terminal of winding 1
N1=	is the three-character keyword "N1=" representing the number of turns in winding 1
t1	is a positive floating-point number assigned as the number of turns in winding 1

For a k-winding transformer, the node names of each winding and the number of turns in each winding must be specified in this device statement.

Simple Switches

The formats for simple switches are:

S*name* *n+* *n-* *nc+* *nc-* *mname* IC={CLOSE|OPEN}

S*name* *n+* *n-* *cname* *mname* IC={CLOSE|OPEN}

where

S	is the one-character element keyword "S" for simple switches
name	is the individual name of the device
n+	is the name of the positive node of the simple switch and is a nonnegative integer
n-	is the name of the negative node of the simple switch and is a nonnegative integer
nc+	is the name of the positive controlling node
nc-	is the name of the negative controlling node
cname	is the name of a controlling device
mname	is the name of a compatible switch model
IC=	is the three-character keyword "IC=" representing the initial condition of the simple switch
OPEN	is the four-character keyword "OPEN", meaning the simple switch is initialized to the open state
CLOSE	is the five-character keyword "CLOSE", meaning the simple switch is initialized to the closed state (use of OPEN and CLOSE are mutually exclusive)

The parameters describing the switch are defined in a model statement. Refer to "Simple Switch Models" on page 57 for the explanation of the model statements associated with simple switches.

In SIMPLIS, both the voltage-controlled and the current-controlled switches are modeled by the simple S switch. If the switch model named "*mname*" has a model type of VCSW, the switch is considered to be voltage controlled. If the model type of the switch model is ICSW, the switch is considered to be current controlled.

The initial condition provided for a simple switch is only used by SIMPLIS as a suggestion. If the circuit condition on the controlling variable dictates a different initial condition, SIMPLIS automatically overrides the given initial condition with the correct initial condition.

Simple Transistor Switches

The formats for simple transistor switches are:

Q*name* *n+* *n-* *nc+* *nc-* *mname* IC={CLOSE|OPEN}

Q*name* *n+* *n-* *cname* *mname* IC={CLOSE|OPEN}

where

Q	is the one-character element keyword "Q" for simple transistor switches
name	is the individual name of the device
n+	is the name of the positive node of the simple transistor switch and is a nonnegative integer
n-	is the name of the negative node of the simple transistor switch and is a nonnegative integer
nc+	is the name of the positive controlling node
nc-	is the name of the negative controlling node
cname	is the name of a controlling device
mname	is the name of a compatible transistor switch model
IC=	is the three-character keyword "IC=" representing the initial condition of the simple transistor switch
OPEN	is the four-character keyword "OPEN", meaning the simple transistor switch is initialized to the open state
CLOSE	is the five-character keyword "CLOSE", meaning the simple transistor switch is initialized to the closed state

The parameters describing the transistor switch are defined in a model statement. Refer to "Simple Switch Models" on page 57 for the explanation of the model statements associated with simple transistor switches.

There are four model types compatible with simple transistor switches. These are:

VCQPOS,
VCQNEG,
ICQPOS, and
ICQNEG.

Model types VCQPOS and VCQNEG correspond to voltage-controlled transistor switches. Model types ICQPOS and ICQNEG correspond to current-controlled transistor switches.

Similar to the initial condition given to a simple switch, the initial condition given to a simple transistor switch is used by SIMPLIS only as a suggestion. Giving a correct initialization, however, eliminates the computation time required by SIMPLIS to search for the correct initial state, and thus leads to a faster simulation.

Piecewise Linear Resistors

The format for piecewise-linear resistor is:

!R*name* n+ n- mname IC=seg_num

where

!R	is the two-character element keyword "!R" for piecewise-linear

	resistors
name	is the individual name of the device
n+	is the name of the positive node and is a nonnegative integer
n-	is the name of the negative node and is a nonnegative integer
mname	is the name of a model compatible with a piecewise-linear resistor
IC=	is the three-character keyword "IC=" representing the initial segment of operation for the piecewise-linear resistor
seg_num	is a positive integer assigned as the initial segment of operation for the piecewise-linear resistor. It must be larger than or equal to 1, and less than or equal to the number of segments defined in the model.

The parameters describing a piecewise-linear resistor are defined in a model statement. Refer to "Piecewise-Linear Resistor Models" on page 52 for the explanation of the model statements associated with the piecewise-linear resistors.

There are two types of models that are compatible with piecewise-linear resistors: VPWLR for voltage-defined piecewise-linear resistors and IPWLR for current-defined piecewise-linear resistors.

The initial segment of operation is used by SIMPLIS as a suggestion. SIMPLIS automatically computes the circuit voltages and currents to determine the correct initial segment of operation.

Piecewise-Linear Inductors and Capacitors

The formats for piecewise-linear inductors and piecewise-linear capacitors are:

!L*name* *n+* *n-* *mname* IC=*init_cond*

!C*name* *n+* *n-* *mname* IC=*init_cond*

where

!L	is the two-character element keyword "!L" for piecewise-linear inductors
!C	is the two-character element keyword "!C" for piecewise-linear capacitors
name	is the individual name of the device
n+	is the name of the positive node and is a nonnegative integer
n-	is the name of the negative node and is a nonnegative integer
mname	is the name of a model compatible with a piecewise-linear inductor or a piecewise-linear capacitor
IC=	is the three-character keyword "IC=" representing the initial condition

init_cond	is the value of initial condition. It is the initial current (in amperes) in the case of a piecewise-linear inductor and the initial voltage (in volts) in the case of a piecewise-linear capacitor.

The parameters describing the piecewise-linear inductors and capacitors are defined in a model statement. Refer to "Piecewise-Linear Inductor and Capacitor Models" on page 55 for the explanation of the model statements associated with these two devices. For a piecewise-linear inductor, the only acceptable model type is PWLL. For a piecewise-linear capacitor, the only acceptable model type is PWLC.

Simple Logic Gates

The format for a simple logic gate is:

!D*name* *no1* *no2* ... *nref* *ni1* *ni2* ... *mname* IC={0|1}

where

!D	is the two-character element keyword "!D" for simple logic gates
name	is the individual name of the device
no1	is a nonnegative integer representing the name of the first output node of the logic gate
no2	is a nonnegative integer representing the name of the second output node of the logic gate if there is more than one output
nref	is a nonnegative integer representing the name of the reference node. The logic state(s) of the output(s) of a simple logic gate are defined in terms of the voltage(s) of the output node(s) with respect to the reference node. The logic state(s) of the input(s) of a simple logic gate are defined in terms of the voltage(s) of the input node(s) with respect to the reference node
ni1	is a nonnegative integer representing the name of the first input node of the logic gate
ni2	is a nonnegative integer representing the name of the second input node of the logic gate if there are more than one input
mname	is the name of a model compatible with a simple logic gate
IC=	is the three-character keyword "IC=" representing the initial output state of the logic gate
0	is the integer 0 to indicate that the initial output state of this gate is logic 0
1	is the integer 1 to indicate that the initial output state of this gate is logic 1 (use of the 0 and 1 are mutually exclusive)

The parameters describing the simple logic gates are defined in a model statement. Refer to "Simple Logic Device Models" on page 64 for the explanation of the model statements associated with simple logic gates. The initial output state of the logic gate is used by SIMPLIS as a suggestion.

The model types and the associated simple logic gates are summarized in the table below.

Model Type	Gate Type	Num. Inputs	Num. Outputs
INV	Inverter	1	1
COMP	Comparator	2	1
XOR	Exclusive OR gate	2	1
ORk	k-input OR gate $2 \le k \le 9$	k	1
NORk	k-input NOR gate $2 \le k \le 9$	k	1
ANDk	k-input AND gate $2 \le k \le 9$	k	1
NANDk	k-input NANDk gate $2 \le k \le 9$	k	1
SRFF	Set-Reset Flip Flop	2	2
CLK_SRFF	Clocked Set-Reset Flip Flop	3	2
CLK_JKFF	Clocked JK Flip Flop	3	2
CLK_DFF	Clocked D Flip Flop	2	2
CLK_TFF	Clocked Toggle Flip Flop	2	2
LATCH	Latch	2	1

SIMPLIS Digital Model Types

Subcircuit Calls / Instantiation

The format for a subcircuit call is:

X*name n1 n2 ... nn sname*

where

X	is the one-character element keyword "X" for the subcircuit call/instantiation
name	is the individual name of the device
n1	is a nonnegative integer to represent the name of the first node of this device
n2	is a nonnegative integer to represent the name of the second node of this device

nn is the nonnegative integer representing the nth node of this device, and

sname is the name of a subcircuit definition compatible with this device.
Through the use of the subcircuit feature, one can model an n-terminal physical device by building an n-terminal subcircuit made up of the simple basic devices outlined in this section. The subcircuit feature is further explained in See “Subcircuit Definition” on page 86.

Chapter 4 Model Statements

Overview

For a simple device, the number of parameters required to model the device is relatively small and the parameters can be easily blended with the device statement. For example, the resistance, the inductance, and the capacitance of a linear resistor, inductor, and capacitor, respectively, are all defined in the device statements. For devices such as simple switches, piecewise-linear elements, and simple logic gates, a large number of parameters is needed to describe the device performance. In such cases, the model statements provide a convenient and organized way to define the model parameters.

There are two additional advantages in using the model statements. Quite often, several devices in the system being studied may have the same model parameters. In such cases, one single model statement can provide the model parameters for all of the devices of the same type. Another benefit of this arrangement is when several devices are described by the same device model, then the model characteristics of all of these devices can be altered at the same location by modifying the model statement which is common to all.

A typical model statement can be represented by the following example statement:

```
.MODEL   mname   mtype   param   param ...
```

where

.MODEL	is the six-character keyword ".MODEL"
mname	is a legal model name as explained in “Model Names and Subcircuit Names” on page 12. A model name must be unique within a general circuit. If a name is used as a model name in a general circuit, it cannot be used as a subcircuit name in the same general circuit and vice versa.
mtype	is a keyword which stands for one of the model types supported by SIMPLIS. The list of device models recognized by SIMPLIS is shown in the table below.
param	is a parameter assignment in the form illustrated in “Parameter Assignments” on page 21

Model Type	Description
VCSW	Voltage-Controlled Simple Switch
ICSW	Current-Controlled Simple Switch
VCQPOS VCQNEG	Voltage-Controlled Simple Transistor Switch
ICQPOS ICQNEG	Current-Controlled Simple Transistor Switch
VPWLR IPWLR	Piecewise-Linear Resistor
PWLL	Piecewise-Linear Inductor
PWLC	Piecewise-Linear Capacitor
INV	Simple Logic Gate, Inverter
COMP	Simple Logic Gate, Comparator
XOR	Simple Logic Gate, Exclusive OR
ORk	Simple Logic Gate, k-input OR, where k is an integer, $2 \le k \le 9$
NORk	Simple Logic Gate, k-input NOR, where k is an integer, $2 \le k \le 9$
ANDk	Simple Logic Gate, k-input AND, where k is an integer, $2 \le k \le 9$
NANDk	Simple Logic Gate, k-input NAND, where k is an integer, $2 \le k \le 9$
SRFF	Simple Logic Gate, Set-Reset Flip Flop
CLK_SRFF	Clocked Logic Gate, Clocked Set-Reset Flip Flop
CLK_JKFF	Clocked Logic Gate, Clocked JK Flip Flop
CLK_DFF	Clocked Logic Gate, Clocked D Flip Flop
CLK_TFF	Clocked Logic Gate, Clocked Toggle Flip Flop
LATCH	Latch

Device Model Types Used by SIMPLIS

The keyword ".MODEL", the model name, and the model type must be entered in the exact order as indicated. Following these three fields are a number of fields each made up of a parameter assignment. The actual number of parameters assignments depends on the model type. The number of parameter assignments must be exactly equal to what is required by the model type. Extra or missing parameter assignments will lead to error messages. Within the set of fields for the parameter assignments, however, the fields can appear in any order of sequence. For example, the following two statements are both acceptable:

```
.MODEL S1 VCSW TH=2 HYSTWD=2U RON=10m
+ ROFF=10MEG LOGIC=POS
```

or

```
.MODEL S1 VCSW RON=10m ROFF=10MEG
+ TH=2 HYSTWD=2U LOGIC=POS
```

Parameter assignments in SIMPLIS do not assume default values. Therefore, each required parameter must be assigned a proper value in the .MODEL statement.

Device Models Used in Simplis

Piecewise-Linear Resistor Models

There are two acceptable model types for piecewise-linear resistors:

1. A voltage-defined piecewise-linear resistor, VPWLR
2. A current-defined piecewise-linear resistor, IPWLR

The formats for the model statements associated with these two model types are:

.MODEL *mname mtype* NSEG=*k* X0=*x0* Y0=*y0*
+ X1=*x1* Y1=*y1* X2=*x2* Y2=*y2* ... Xk=*xk* Yk=*yk*

where

.MODEL	is the six-character keyword ".MODEL"
mname	is a legal model name as explained in "Model Names and Subcircuit Names" on page 12
mtype	is a five-character keyword equal to either "VPWLR" or "IPWLR"
NSEG=	is the five-character keyword "NSEG=" representing the number of linear segments in the resistor model
k	is an integer which defines the number of linear segments for this piecewise-linear resistor and can take on values from 2 to 255, inclusively
X0=	is the keyword "X0=" representing the voltage (x axis) coordinate of the beginning of the first linear segment of the piece-wise linear resistor
x0	is a floating-point number which defines the value of X0 in volts.
Y0=	is the keyword "Y0=" representing the current (y axis) coordinate of the beginning of the first linear segment of the piece-wise linear resistor
y0	is a floating-point number that defines the value of Y0 in amperes, so that the straight line passing through (x0, y0) and terminating on the break point (x1, y1) forms the first segment of the piecewise-linear characteristic
X1=	is the keyword "X1=" representing the voltage (x axis) coordinate of the end of the first linear segment of the piece-

	wise linear resistor and the beginning of the second linear segment of the resistor
x1	is a floating-point number which defines the value of X1 in volts.
Y1=	is the keyword "Y1=" representing the current (y axis) coordinate of the end of the first linear segment of the piece-wise linear resistor and the beginning of the second linear segment of the resistor
y1	is a floating-point number which defines the value of Y1 in amperes.
X2=	is the keyword "X2=" representing the voltage (x axis) coordinate of the end of the second linear segment of the piece-wise linear resistor and the beginning of the third linear segment of the resistor
x2	is a floating-point number which defines the value of X2 in volts.
Y2=	is the keyword "Y2=" representing the current (y axis) coordinate of the end of the second linear segment of the piece-wise linear resistor and the beginning of the third linear segment of the resistor
y2	is a floating-point number which defines the value of Y2 in amperes, so that the straight line starting at the break point (x1, y1) and terminating at the break point (x2, y2) forms the second segment of the piecewise-linear characteristic, and so on
Xk=	is the keyword "Xk=" representing the voltage (x axis) coordinate of the end of the kth (last) linear segment of the piece-wise linear resistor
xk	is a floating-point number which defines the value of Xk in volts.
Yk=	is the keyword "Yk=" representing the current (y axis) coordinate of the end of the kth (last) linear segment of the piece-wise linear resistor

yk is a floating-point number which defines the value of Yk in amperes, so that the straight line starting at the break point (*xk*-1, *yk*-1) and passing through the point (*xk*, *yk*) forms the last segment of the piecewise-linear characteristic.

The slope of each line segment on the v-i plane is the differential conductance in Siemens for the device. The small-signal resistance is then the reciprocal of the differential conductance.

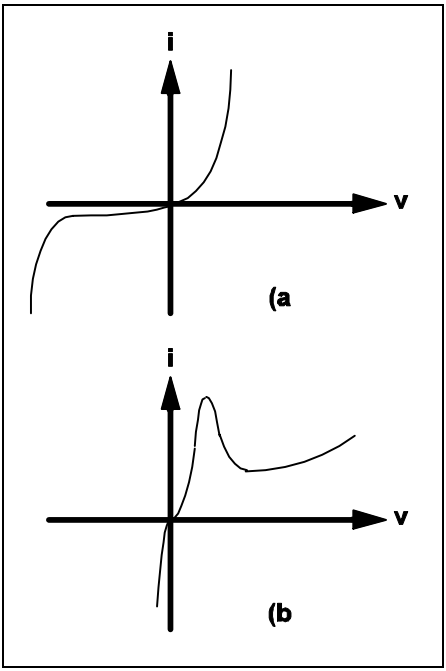
VPWLR-Type Model

The VPWLR model type is used for piecewise-linear v-i characteristics which are voltage-defined. In other words, the value of current is uniquely defined for every value of voltage. In such a case, the voltage-current coordinates at the points of definition of the v-i characteristics must satisfy the following two restrictions:

1. Values of the voltages must be entered in a strictly ascending order:
   x0 < x1 < x2 < ... < xk
2. The slopes of the first and last segments must be positive:
   y0 < y1 and yk-1 < yk

Other than the first and the last segments, each intermediate segment j has two break points and is defined for voltages in the range of $xj\text{-}1 \le v \le xj$. The first segment has one break point at ($x1$, $y1$) and is defined for voltages in the range of $v \le x1$. The last segment has one break point at (xk-1, yk-1) and is defined for voltages in the range of $xk\text{-}1 \le v$. As such, ($x0$, $y0$) and (xk, yk) are used to define the slopes of the first and last segments instead of being used to define their break points.

As an example, the diagram below shows the v-i characteristics of an ordinary pn-junction diode and that of a tunnel diode. It is apparent from the diagram that the characteristics of these two devices satisfy the voltage-defined requirement and each of these two devices can be modeled by a VPWLR-type piecewise-linear resistor.



4.1 The v-i characteristics of (a) an ordinary pn-junction diode, and (b) a tunnel diode

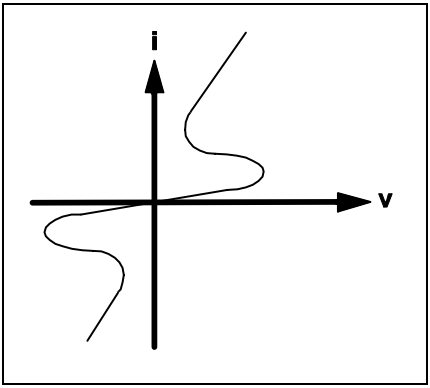
IPWLR-Type Model

The IPWLR model type is reserved for piecewise-linear v-i characteristics which are current-defined in the sense that the value of voltage is uniquely defined for every value of current. In such a case, the voltage-current coordinates at the points of definition of the v-i characteristics must satisfy the following restrictions:

1. Values of the currents must be entered in a strictly ascending order:
   y0 < *y1* < *y2* ... < *yk*
2. The slopes of the first and last segments must be positive:
   x0 < *x1* and *xk* -1 < *xk*

Other than the first and the last segments, each intermediate segment j has two break points and is defined for currents in the range of yj -1 ≤ i ≤ yj. The first segment has one break point at (*x1*, *y1*) and is defined for currents in the range of i ≤ y1. The last segment has one break point at (*xj*-1, *yj*-1) and is defined for currents in the range of *i* *yk* - 1. The two points (*x0*, *y0*) and (*xk*, *yk*) are used in conjunction with (*x1*, *y1*) and (*xk*-1, *yk*-1) to define the slopes of the first and the last segments of the characteristics.

For example, the v-i characteristics of the ordinary pn-junction diode shown in diagram 4.1(a) above and the v-i characteristics shown in diagram 4.2 can both be considered to behave as nonlinear current-defined resistors. Therefore, each can be approximated by an IPWLR-type model. On the other hand, the tunnel diode characteristics shown in diagram 4.1(b) cannot be modeled by an IPWLR-type model since the values for the branch voltage are not uniquely defined for every value of current. Similarly, the v-i characteristics shown in 4.2 cannot be modeled by a VPLWR-type model.



4.2 Example of a type of v-i characteristics which can be described as a current-defined resistor

Piecewise-Linear Inductor and Capacitor Models

The model statements for piecewise-linear inductors and capacitors are very similar to those for the piecewise-linear resistors. In the case of a piecewise-linear resistor, its characteristics are defined in terms of points on the current vs voltage plane. In the case of a piecewise-linear inductor, the characteristics are defined in terms of points on the flux-linkage vs. current plane. In the case of a piecewise-linear capacitor, the characteristics are defined in terms of points on the charge vs voltage plane.

The model statement format for a piecewise-linear inductor or capacitor is:

.MODEL *mname* *mtype* NSEG=*k* X0=*x0* Y0=*y0*

+ X1=*x1* Y1=*y1* X2=*x2* Y2=*y2* ... Xk=*xk* Yk=*yk*

where

.MODEL	is the six-character keyword ".MODEL"
mname	is a legal model name as explained in "Model Names and Subcircuit Names" on page 12
mtype	is a four-character keyword equal to "PWLL" or "PWLC", which stand for the model types for a piecewise-linear inductor or capacitor, respectively
NSEG=	is the five-character keyword "NSEG=" representing the number of linear segments in this device model
k	is an integer defining the number of linear segments and can take on values from 2 to 255, inclusively
X0=	is the keyword "X0=" representing the x axis coordinate of the beginning of the first linear segment of the device model
x0	is a floating-point number which defines the value of X0 and has units of amperes for a PWL inductor and volts for a PWL capacitor.
Y0=	is the keyword "Y0=" representing the y axis coordinate of the beginning of the first linear segment of the device model
y0	is a floating-point number which defines the value of Y0 and has units of weber-turns for a PWL inductor and coulombs for a PWL capacitor
X1=	is the keyword "X1=" representing the x axis coordinate of the end of the first linear segment of the device model and the beginning of the second linear segment of the device model
x1	is a floating-point number which defines the value of X1 and has the same units indicated for x0
Y1=	is the keyword "Y1=" representing the y axis coordinate of the end of the first linear segment of the device model and the beginning of the second linear segment of the device model, so that the straight line passing through (x0, y0) and terminating on the break point (x1, y1) forms the first segment of the piecewise-linear characteristic
y1	is a floating-point number which defines the value of Y1
X2=	is the keyword "X2=" representing the x axis coordinate of the end of the second linear segment of the device model and the beginning of the third linear segment of the device model
x2	is a floating-point number which defines the value of X2.
Y2=	is the keyword "Y2=" representing the y axis coordinate of the end of the second linear segment of the device model and the beginning of the third linear segment of the device model, so that the straight line passing through (*x1, y1*) and terminating on

	the break point (*x2*, *y2*) forms the second segment of the piecewise-linear characteristic
y2	is a floating-point number which defines the value of Y2, and so on
Xk=	is the keyword "Xk=" representing the x axis coordinate of the end of the last linear segment of the device model
xk	is a floating-point number which defines the value of Xk.
Yk=	is the keyword "Yk=" representing the y axis coordinate of the end of the last linear segment of the device model, so that the straight line starting at the break point (*xk*-1, *yk*-1) and passing through the point (*xk*, *yk*) forms the last segment of the piecewise-linear characteristic, and
yk	is a floating-point number which defines the value of Yk.

For a PWL inductor, the slope of each line segment on the flux-linkage vs current plane is the differential inductance of the device in Henries. For a PWL capacitor, the slope of each line segment on the charge vs voltage plane is the differential capacitance of the device in Farads. To ensure that the differential inductance or differential capacitance is positive to reflect the characteristics of realistic devices, the following additional restrictions are placed on the values of the coordinate pairs:

x0 < x1 < x2 < ... < xk

y0 < y1 < y2 < ... < yk

Simple Switch Models

SIMPLIS accepts two types of simple switch models:

1. model type VCSW for a voltage-controlled switch, and
2. model type ICSW for a current-controlled switch.

The formats for both of these two model types are:

.MODEL *mname mtype* RON=*ron* ROFF=*roff*
+ TH=*threshold* HYSTWD=*hystwd* LOGIC={POS|NEG}

where

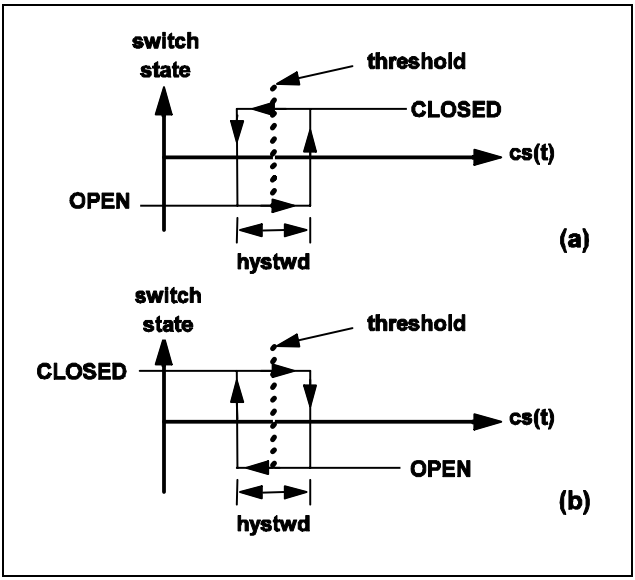
.MODEL	is the six-character keyword ".MODEL"
mname	is a legal model name as explained in “Model Names and Subcircuit Names” on page 12
mtype	is a four-character keyword equal to either "VCSW" or "ICSW", indicating whether the switch is voltage-controlled or current-controlled
RON=	is the four-character keyword "RON=" representing the resistance in ohms of the switch when it is in the closed, or on, state

ron	is a positive floating-point number which defines the resistance in ohms of the switch when it is in the closed, or on, state
ROFF=	is the five-character keyword "ROFF=" representing the leakage resistance in ohms of the switch when it is in the open state
roff	is a positive floating-point number which defines the leakage resistance of the switch in ohms when it is at the open state
TH=	is the three-character keyword "TH=" representing the threshold value of the controlling signal. Together with HYSTWD, it determines the values at which the state of the switch will be changed from an open state to a closed state and vice versa
threshold	is a floating-point number which defines the threshold value of the controlling signal and is measured in volts for a voltage-controlled switch and measured in amperes for a current-controlled switch
HYSTWD=	is the seven-character keyword "HYSTWD=" representing the hysteresis width of the controlling signal
hystwd	is a positive floating-point number which defines the hysteresis width of the controlling signal and has the same unit of measurement as that of threshold.
LOGIC=	is the six-character keyword "LOGIC="
POS	is the three-character string "POS"
NEG	is the three-character string "NEG".

If the model type is VCSW, the controlling signal cs(*t*) for the simple switch is the voltage of a pair of controlling nodes or the branch voltage of a controlling device. If the model type is ICSW, the controlling signal cs(*t*) for the simple switch is the branch current of a controlling device.

The diagram below defines the state of the simple switch, under two operating modes. If LOGIC is assigned the value POS, the switching of the simple switch is defined by (a). If LOGIC is assigned the value NEG, the switching logic is reversed, and the state of the simple switch is then defined by (b).

When a simple switch is in the closed state, it is modeled by a linear resistor having a resistance equal to ron between its positive node and negative node. When the simple switch is in the open state, the resistance of the linear resistor changes to roff.



4.3 State diagram of the simple switch when the parameter LOGIC is assigned a value of (a) POS or (b) NEG

Simple Transistor Switch Models

There are four simple transistor models, composed of:

1. a set of two voltage-controlled models, designated as VCQPOS and VCQNEG;
2. a set of two current-controlled models, designated as ICQPOS and ICQNEG.

The formats for these four model types are:

.MODEL *mname mtype* VSAT=*vsat* RSAT=*rsat*
+ ROFF=*roff* GAIN=*gain* TH=*threshold*
+ HYSTWD=*hystwd* LOGIC={POS|NEG} LEVEL={1|2}

where

.MODEL	is the six-character keyword ".MODEL"
mname	is a legal model name as explained in "Model Names and Subcircuit Names" on page 12
mtype	is a six-character keyword equal to one of the following keywords: "VCQPOS", "VCQNEG", "ICQPOS", and "ICQNEG"
VSAT=	is the five-character keyword "VSAT="
vsat	is a floating-point number which defines the saturation voltage

	of the transistor switch in volts. It is the voltage across the transistor switch when it is saturated and the current through it is negligibly small. It is a positive number for VCQPOS-type and ICQPOS-type transistor switches and it is a negative number for VCQNEG-type and ICQNEG-type transistor switches,
RSAT=	is the five-character keyword "RSAT="
rsat	is a positive number which defines the saturation resistance in ohms of the transistor switch
ROFF=	is the five-character keyword "ROFF="
roff	is a positive number which defines the leakage resistance of the switch in ohms when it is at the open state. It must be larger than *rsat*
GAIN=	is the five-character keyword "GAIN="
gain	is a positive floating-point number which defines the gain of the transistor switch when the parameter LEVEL is assigned a value above 1. Its value is ignored when LEVEL is assigned a value of 1
TH=	is the three-character keyword "TH="
threshold	is a floating-point number which defines the threshold value of the controlling signal. Together with hystwd, it determines the values at which the transistor switch will be changed from an OPEN state to a CLOSE state and vice versa. It is measured in volts for VCQPOS- and VCQNEG-type switches and measured in amperes for ICQPOS- and ICQNEG-type switches
HYSTWD=	is the seven-character keyword "HYSTWD="
hystwd	is a positive floating-point number which defines the hysteresis width of the controlling signal. It has the same unit of measurement as that of threshold
LOGIC=	is the six-character keyword "LOGIC="
POS	is the three-character keyword "POS"
NEG	is the three-character keyword "NEG",
LEVEL=	is the six-character keyword "LEVEL="
1	is the integer 1
2	is the integer 2

If the model type is VCQPOS or VCQNEG, the controlling signal cs(*t*) for the simple transistor switch is the voltage of a pair of controlling nodes or the branch voltage of a controlling device. If the model type is ICQPOS or ICQNEG, the controlling signal cs(*t*) for the simple transistor switch is the branch current of a controlling device.

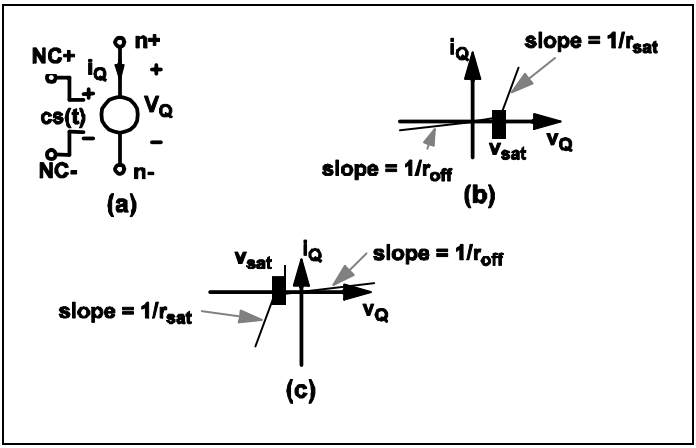
For model types VCQPOS and ICQPOS, the voltage across the transistor switch, measured as the voltage of the positive node with respect to the voltage of the negative node, assumes nonnegative values under normal operation like an NPN bipolar transistor and an N-channel MOSFET. For model types VCQNEG and ICQNEG, the

voltage across the transistor switch, measured as the voltage of the positive node with respect to the voltage of the negative node, assumes non positive values under normal operation like a PNP bipolar transistor and a P-channel MOSFET.

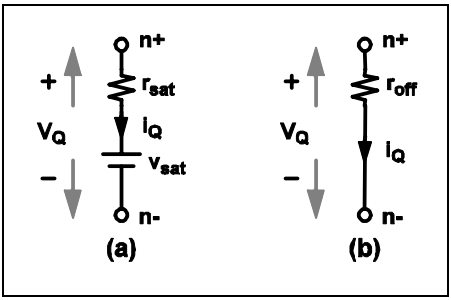
Models for Simple Transistor Switches for LEVEL=1

When a simple transistor switch is modeled with the LEVEL parameter set to 1, it can assume either an open or a closed state. The switching diagram in 4.3 (a) applies when LOGIC is set to POS while the switching diagram in 4.3 (b) applies when LOGIC is set to NEG.

The only difference between a simple switch and a simple transistor switch with LEVEL set to 1 is the model of the closed state. When a simple transistor switch with LEVEL set to 1 is in the closed state, it is modeled by a linear resistor with the value *rsat* in series with a constant voltage source with the value *vsat*. When the simple transistor switch is in the open state, it is modeled by a resistor with the value *roff*. The block diagram and the V-I characteristic of the model are shown in Figure 4.4 The circuit element model of the simple transistor switch with LEVEL set to 1 for the closed and open states are shown in diagram 4.4 below.



4.4 Model for the simple transistor switch: (a) Simple transistor switch controlled by a control signal cs(t), (b) the i Q vs v Q characteristic of the simple POS-type transistor switch, and (c) the i Q vs v Q characteristic of the simple NEG-type transistor switch



4.5 Model for the simple transistor switch: (a) model of the simple transistor switch for LEVEL=1 when it is in a closed state, and (b) model of the simple transistor switch for LEVEL=1 when it is in an open state.

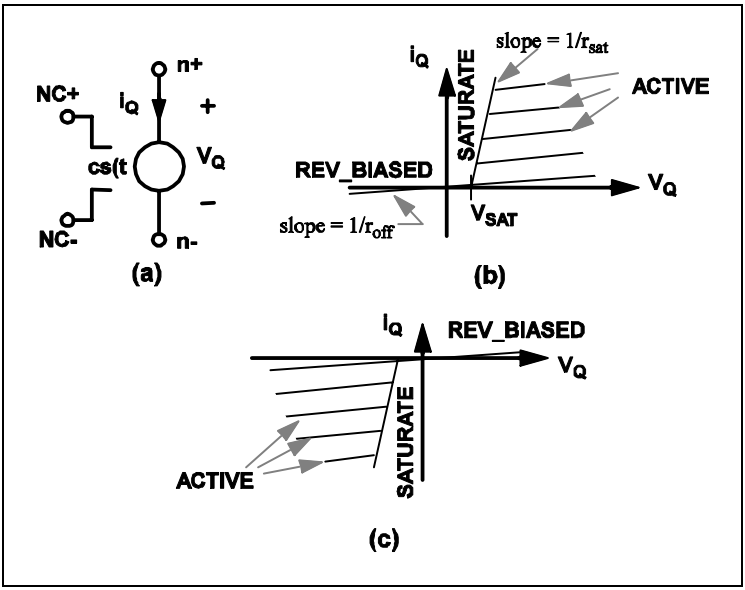
If a physical transistor is being driven to act like a switch and detailed waveforms of the voltage across the transistor and the current through the transistor are not critically important, it is recommended that such a transistor be modeled by a simple transistor switch with the parameter LEVEL set to 1 since the simulation is faster when a transistor switch has LEVEL set to 1.

When a simple transistor switch is modeled with the LEVEL parameter set to 1, the value of the GAIN parameter has no effect on the modeling. In addition, the direction of current flow through the transistor switch is not restricted and the device behaves more like a controlled switch than a physical transistor.

Models for Simple Transistor Switches for LEVEL=2

When a simple transistor switch is modeled with the LEVEL parameter set to 2, it still assumes either an open state or a closed state. The switching diagram in 4.3(a) still applies when LOGIC is set to POS while the switching diagram in 4.3 (b) still applies when LOGIC is set to NEG.

When the LEVEL parameter is set to 2, the simple transistor is still modeled by a resistor with a resistance equal to *roff* when it is in the open state. When the simple transistor switch is in the closed state, additional secondary states are provided for the simple transistor, allowing the modeling of a physical transistor at operating areas where both the voltage across and the current through the transistor are simultaneously substantial. For bipolar transistors, such operating areas are collectively called the active region. For the simple transistor switch the individual states are called ACTIVE, SATURATE, and REV_BIASED, to stand for active region, saturation, and reversed-biased, as indicated in 4.6 . SIMPLIS internally computes the voltage across and the current through the simple transistor switch to determine the correct secondary state at which the transistor switch should operate.



4.6 Model for the simple transistor switch for LEVEL=2. (a) A simple transistor switch controlled by a control signal cs(t), (b) The i Q vs v Q characteristics of a VCQPOS-type or ICQPOS-type transistor switch, (c) The i Q vs v Q characteristic of a VCQNEG-type or ICQNEG-type transistor switch

When the secondary state of a simple transistor switch is equal to ACTIVE, it is modeled by a parallel combination of a linear resistor with a resistance equal to roff and a controlled-current source ci(*t*) as indicated in 4.7(a). The current ci(*t*) of the controlled-current source is defined by the following equations if LOGIC is set to POS:

ci(t) = gain * [cs(t) - (threshold - hystwd/2)]

for VCQPOS-type transistor switches and
for ICQPOS-type transistor switches

AND

ci(t) = - gain * [cs(t) - (threshold - hystwd/2)]

for VCQNEG-type transistor switches and
for ICQNEG-type transistor switches.

On the other hand, the current ci(t) of the controlled-current source is defined by the following equations if LOGIC is set to NEG:

ci(t) = - gain * [cs(t) - (threshold + hystwd/2)]

for VCQPOS-type transistor switches and
for ICQPOS-type transistor switches, and

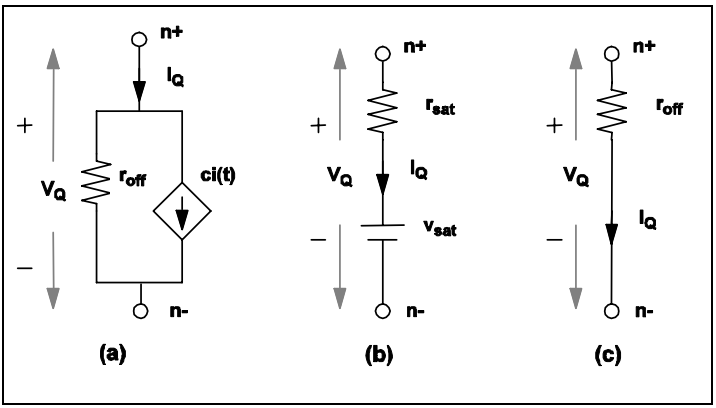
AND

ci(t) = gain * [cs(t) - (threshold + hystwd/2)]

for VCQNEG-type transistor switches and
for ICQNEG-type transistor switches.

For example, an NPN transistor can be modeled by a piecewise-linear resistor to represent the base-emitter characteristics and an ICQPOS-type simple transistor switch with LEVEL and LOGIC set to 2 and POS, respectively, to represent the collector-emitter characteristics. Similarly, the collector-emitter characteristics of a PNP transistor can be modeled by an ICQNEG-type transistor switch with LEVEL and LOGIC set to 2 and NEG, respectively.

When the secondary state of a simple transistor switch is equal to SATURATE, it is modeled by the small network as shown in 4.7(b), which comprises a linear resistor with a resistance equal to *rsat* in series with a voltage source with source value vsat. When the secondary state of a simple transistor switch is equal to REV_BIASED, it is modeled by a large resistor with resistance equal to *roff* as shown in 4.7 (c).



4.7 Model of a simple transistor switch in the closed state, with LEVEL=2. (a) Model for the ACTIVE secondary state, (b) Model for the SATURATE secondary state, (c) Model for the REV_BIASED secondary state

By selecting LEVEL=2 in the simple transistor switch model, you can more accurately model a physical transistor and are able to obtain more detailed waveforms on the voltage and current for the device. The penalty is an increase in the simulation time since more variables need to be monitored and computed throughout the simulation.

Simple Logic Device Models

To aid the understanding of the models for simple logic devices, the concept of "positive" and "negative" logic is discussed first. Except for inverters and comparators,

all logic gates use a parameter called LOGIC in the model statement. The purpose of this parameter is to define whether a "positive logic" or a "negative logic" convention is used in defining the logic states.

If the LOGIC parameter is set to POS, positive logic is used to determine the logic states of the inputs and the output. This means a state of logic 0 is represented by a lower voltage level than that of a state of logic 1. In this case, the input logic state of an input node is defined to be at logic 1 if

$$V(ni,nref) \geq threshold + hystwd/2$$

where *ni* and *nref* are the node names of the input node and reference node, respectively.

The input logic state is considered to be at logic 0 if

$$V(ni,nref) \leq threshold - hystwd/2$$

The output logic state is equal to the result of the boolean operator associated with the gate applied to the input logic states. If the output state is equal to logic 1, the value of the voltage source in the output circuit is set to *voh*. If the output state is equal to logic 0, the value of the voltage source in the output circuit is set to *vol*. The two parameters *vol* and *voh* are specified in the model statement.

If the LOGIC parameter is set to NEG, negative logic is used to determine the logic states of the inputs and the output. Negative logic means a state of logic 0 is represented by a higher voltage level than that of a state of logic 1. In this case, the input logic state of an input node *ni* is defined to be at logic 1 if

$$V(ni,nref) \leq threshold - hystwd/2$$

and it is considered to be at logic 0 if

$$V(ni,nref) \geq threshold + hystwd/2$$

The output logic state is equal to the result of the boolean operator associated with the gate applied to the input logic states. If the output state is equal to logic 1, the value of the voltage source, *vout*, in the output circuit is set to *vol*. If the output state is equal to logic 0, the value of *vout* is set to *voh*.

Inverter Model

The format for the inverter model statement is:

.MODEL *mname* INV TH=*threshold* HYSTWD=*hystwd*
+ VOL=*vol* VOH=*voh* RIN=*rin* ROUT=*rout*

where

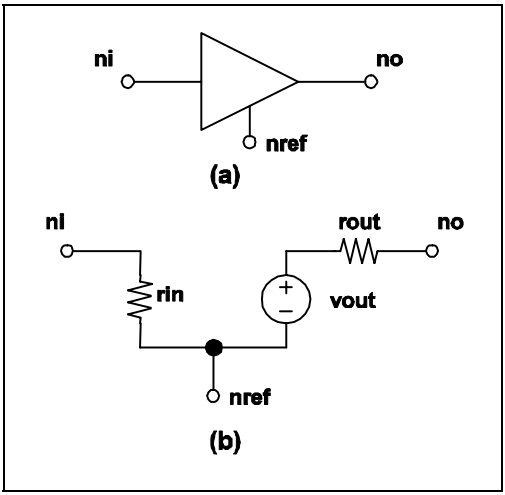
. MODEL is the six-character keyword ".MODEL",

mname is a legal model name as explained in “Model Names and Subcircuit Names” on page 12

INV is the three-character keyword "INV" which identifies the

	inverter-type simple logic gates
TH=	is the three-character keyword "TH="
threshold	is a floating-point number which defines the threshold value of the input voltage in volts. Together with hystwd, it determines the values of the input voltage at which the output states of the inverter will be changed from a logic 0 to a logic 1 and vice versa
HYSTWD=	is the seven-character keyword "HYSTWD="
hystwd	is a positive floating-point number to represent the hysteresis width of the input voltage in volts
VOL=	is the four-character keyword "VOL="
vol	is a floating-point number which defines the low value of the output voltage in volts
VOH=	is the four-character keyword "VOH="
voh	is a floating-point number which defines the high value of the output voltage in volts. It must be larger than *vol*
RIN=	is the four-character keyword "RIN="
rin	is a floating-point number which defines the input resistance in ohms
ROUT=	is the five-character keyword "ROUT="
rout	is a floating-point number which defines the output resistance in ohms.

The actual model implemented in SIMPLIS for an inverter is shown in 4.8 (b). The input circuit is represented by a linear resistor of resistance rin placed between the input and reference nodes. The output circuit is modeled by a Thevenin equivalent network between the output and reference nodes. The value of resistance for the resistor in the Thevenin network is equal to *rout*. The value of the voltage source in the Thevenin network depends on the output state of the inverter.



4.8 SIMPLIS inverter model: (a) Symbol for inverter, (b) Model for inverter. The nodes ni, no and nref are the input, output and the reference nodes, respectively

If the output state of an inverter is equal to logic 1, the value of the voltage source, vout, in the output circuit is set to *voh*. In this case, the output state of the device is changed to logic 0 when

$$V(ni,nref) \geq threshold + hystwd/2$$

where V(*ni*,*nref*) represents the voltage of the input node with respect to the reference node.

If the output state of an inverter is equal to logic 0, the value of the voltage source, vout, in the output circuit is set to vol. In this case, the output state of the device is changed to logic 1 when

$$V(ni,nref) \leq threshold - hystwd/2$$

Comparator Model

The format for the comparator model statement is:

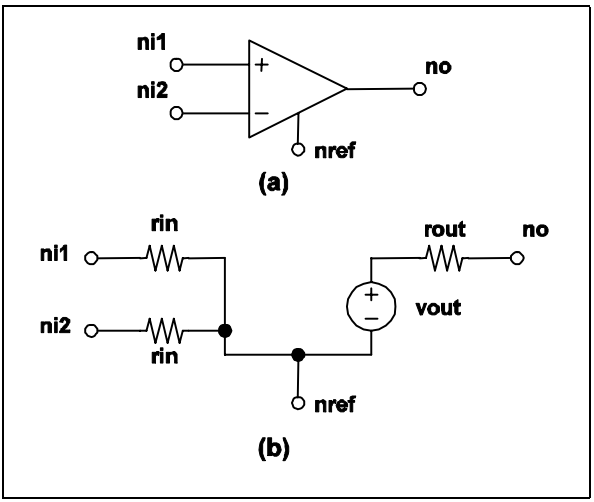
.MODEL *mname* COMP HYSTWD=*hystwd* VOL=*vol* VOH=*voh* + RIN=*rin* ROUT=*rout*

where

.MODEL	is the six-character keyword ".MODEL"
mname	is a legal model name as explained in "Model Names and Subcircuit Names" on page 12

COMP	is the four-character keyword "COMP" to stand for comparator-type simple logic gates
HYSTWD=	is the seven-character keyword "HYSTWD="
hystwd	is a positive floating-point number which defines the hysteresis width of the differential input voltage in volts
VOL=	is the four-character keyword "VOL="
vol	is a floating-point number which defines the low value of the output voltage in volts
VOH=	is the four-character keyword "VOH="
voh	is a floating-point number which defines the high value of the output voltage in volts. It must be larger than *vol*
RIN=	is the four-character keyword "RIN="
rin	is a floating-point number which defines the input resistance in ohms
ROUT=	is the five-character keyword "ROUT="
rout	is a floating-point number which defines the output resistance in ohms.

The actual model implemented in SIMPLIS for a comparator is shown in 4.9 (b). There is a resistor of value rin placed between each input node and the reference node. The output circuit is modeled by a resistor in series with a voltage source. The resistor has a resistance rout and the source value of the voltage source, vout, depends on the logic state of the output of the comparator.



4.9 SIMPLIS comparator model: (a) Symbol for comparator, (b) Model for comparator. The nodes n1, n2 are the two input nodes. The nodes no and nref are the output and reference nodes, respectively

If the output state of a comparator is equal to logic 1, the value of the voltage source, *vout*, in the output circuit is set to *voh*. In this case, the output state of the device is changed to logic 0 when

$$V(ni1,ni2) \leq - hystwd/2$$

where V(*ni1*,*ni2*) represents the voltage of the first input node with respect to the voltage of the second input node.

If the output state of a comparator is equal to logic 0, the value of the voltage source, *vout*, in the output circuit is set to *vol*. In this case, the output state of the device is changed to logic 1 when

$$V(ni1,ni2) \geq hystwd/2$$

Exclusive-OR Gate Model

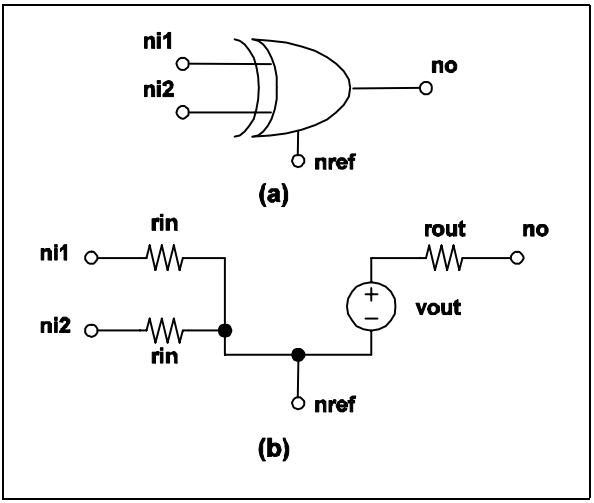
The format for the Exclusive-OR model statement is:

.MODEL *mname* XOR TH=*threshold* HYSTWD=*hystwd*
+ VOL=*vol* VOH=*voh* RIN=*rin* ROUT=*rout*
+ LOGIC={POS | NEG}

where

.MODEL	is the six-character keyword ".MODEL"
mname	is a legal model name as explained in “Model Names and Subcircuit Names” on page 12
XOR	is the three-character keyword "XOR" to stand for exclusive-OR type simple logic gates
TH=	is the three-character keyword "TH="
threshold	is a floating-point number which defines the threshold value of the input voltage in volts. Together with hystwd, it determines the values of the input voltage at which the input states of the exclusive-OR gate will be changed from a logic 0 to a logic 1 and vice versa
HYSTWD=	is the seven-character keyword "HYSTWD="
hystwd	is a positive floating-point number which defines the hysteresis width of the input voltage in volts
VOL=	is the four-character keyword "VOL="
vol	is a floating-point number representing the low value of the output voltage in volts
VOH=	is the four-character keyword "VOH="
voh	is a floating-point number which defines the high value of the output voltage in volts. It must be larger than the value of *vol*
RIN=	is the four-character keyword "RIN="

rin	is a floating-point number which defines the input resistance in ohms
ROUT=	is the five-character keyword "ROUT="
rout	is a floating-point number which defines the output resistance in ohms
LOGIC=	is the six-character keyword "LOGIC="
POS	is the three-character keyword "POS"
NEG	is the three-character keyword "NEG"



4.10 Exclusive-OR gate model: (a) Symbol for exclusive-OR gate, (b) Model for exclusive-OR gate. The nodes ni1 and ni2 are the two input nodes. The nodes no and nref are the output and reference nodes, respectively.

The actual model implemented in SIMPLIS for an exclusive-OR gate is shown in 4.10 (a). The source value of the voltage source, vout, in the output circuit depends on the logic state of the output of the gate. The output state is equal to the result of the boolean EXCLUSIVE-OR operation on the two input states.

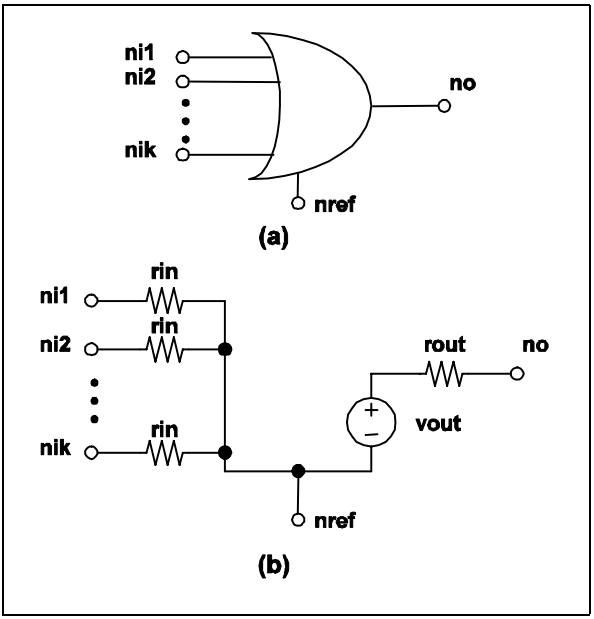
OR Gate Model

The format for the OR Gate model statement is:

.MODEL *mname* ORk TH=*threshold* HYSTWD=*hystwd*
+ VOL=*vol* VOH=*voh* RIN=*rin* ROUT=*rout*
+ LOGIC={POS | NEG}

where

.MODEL	is the six-character keyword ".MODEL"
mname	is a legal model name as explained in “Model Names and Subcircuit Names” on page 12
OR	is the two-character keyword "OR" to stand for OR-type simple logic gates
k	is an integer from 2 to 9, inclusively, which defines the number of inputs for the OR gate
TH=	is the three-character keyword "TH="
threshold	is a floating-point number which defines the threshold value of the input voltage in volts, and together with hystwd, it determines the values of the input voltage at which the input states of the exclusive-OR gate will be changed from a logic 0 to a logic 1 and vice versa
HYSTWD=	is the seven-character keyword "HYSTWD="
hystwd	is a positive floating-point number which defines the hysteresis width of the input voltage in volts
VOL=	is the four-character keyword "VOL="
vol	is a floating-point number representing the low value of the output voltage in volts
VOH=	is the four-character keyword "VOH="
voh	is a floating-point number which defines the high value of the output voltage in volts. It must be larger than the value of *vol*
RIN=	is the four-character keyword "RIN="
rin	is a floating-point number which defines the input resistance
ROUT=	is the five-character keyword "ROUT="
rout	is a floating-point number which defines the output resistance in ohms
LOGIC=	is the six-character keyword "LOGIC="
POS	is the three-character keyword "POS"
NEG	is the three-character keyword "NEG"



4.11 k-Input OR gate model: (a) Symbol for k-Input OR gate, (b) Model for k-Input OR gate. The nodes ni1 and ni2 are the two input nodes. Up to a maximum of 9 inputs can be accommodated

The actual model implemented in SIMPLIS for a k-input OR gate is shown in 4.11 (b). The output state is equal to the result of the boolean OR operation applied to the k input states.

NOR Gate Model

The format for the NOR Gate model statement is:

.MODEL *mname* NORk TH=*threshold* HYSTWD=*hystwd*
+ VOL=*vol* VOH=*voh* RIN=*rin* ROUT=*rout*
+ LOGIC={POS | NEG}

where

.MODEL	is the six-character keyword ".MODEL"
mname	is a legal model name as explained in "Model Names and Subcircuit Names" on page 12
NOR	is the three-character keyword "NOR" to stand for NOR-type simple logic gates
k	is an integer from 2 to 9, inclusively, indicating the number of inputs for the NOR gate

TH=	is the three-character keyword "TH="
threshold	is a floating-point number which defines the threshold value of the input voltage in volts, and together with hystwd, it determines the values of the input voltage at which the input states of the exclusive-OR gate will be changed from a logic 0 to a logic 1 and vice versa
HYSTWD=	is the seven-character keyword "HYSTWD="
hystwd	is a positive floating-point number which defines the hysteresis width of the input voltage in volts
VOL=	is the four-character keyword "VOL="
vol	is a floating-point number representing the low value of the output voltage in volts
VOH=	is the four-character keyword "VOH="
voh	is a floating-point number which defines the high value of the output voltage in volts and must be larger than *vol*
RIN=	is the four-character keyword "RIN="
rin	is a floating-point number which defines the input resistance
ROUT=	is the five-character keyword "ROUT="
rout	is a floating-point number which defines the output resistance in ohms
LOGIC=	is the six-character keyword "LOGIC="
POS	is the three-character keyword "POS"
NEG	is the three-character keyword "NEG"

The output state is equal to the result of the boolean NOR operation applied to the k input states.

AND Gate Model

The format for the AND Gate model statement is:

.MODEL *mname* ANDk TH=*threshold* HYSTWD=*hystwd*
+ VOL=*vol* VOH=*voh* RIN=*rin* ROUT=*rout*
+ LOGIC={POS | NEG}

where

.MODEL	is the six-character keyword ".MODEL"
mname	is a legal model name as explained in "Model Names and Subcircuit Names" on page 12
AND	is the three-character keyword "AND" to stand for AND-type simple logic gates
k	is an integer from 2 to 9, inclusively, to stand for the number of

	inputs for the AND gate
TH=	is the three-character keyword "TH="
threshold	is a floating-point number which defines the threshold value of the input voltage in volts, which together with hystwd, determines the values of the input voltage at which the input states of the exclusive-OR gate will be changed from a logic 0 to a logic 1 and vice versa
HYSTWD=	is the seven-character keyword "HYSTWD="
hystwd	is a positive floating-point number which defines the hysteresis width of the input voltage in volts
VOL=	is the four-character keyword "VOL="
vol	is a floating-point number representing the low value of the output voltage in volts
VOH=	is the four-character keyword "VOH="
voh	is a floating-point number which defines the high value of the output voltage in volts and must be larger than *vol*
RIN=	is the four-character keyword "RIN="
rin	is a floating-point number which defines the input resistance
ROUT=	is the five-character keyword "ROUT="
rout	is a floating-point number which defines the output resistance in ohms
LOGIC=	is the six-character keyword "LOGIC="
POS	is the three-character keyword "POS"
NEG	is the three-character keyword "NEG"

The output state is equal to the result of the boolean AND operation applied to the k input states.

NAND Gate Model

The format for the NAND gate model statement is:

.MODEL *mname* NANDk TH=*threshold* HYSTWD=*hystwd*
+ VOL=*vol* VOH=*voh* RIN=*rin* ROUT=*rout*
+ LOGIC={POS | NEG}

where

.MODEL	is the six-character keyword ".MODEL"
mname	is a legal model name as explained in "Model Names and Subcircuit Names" on page 12
NAND	is the four-character keyword "NAND" to stand for NAND-type simple logic gates

k	is an integer from 2 to 9, inclusively, to stand for the number of inputs for the NAND gate
TH=	is the three-character keyword "TH="
threshold	is a floating-point number which defines the threshold value of the input voltage in volts, which together with hystwd, determines the values of the input voltage at which the input states of the exclusive-OR gate will be changed from a logic 0 to a logic 1 and vice versa
HYSTWD=	is the seven-character keyword "HYSTWD="
hystwd	is a positive floating-point number which defines the hysteresis width of the input voltage in volts
VOL=	is the four-character keyword "VOL="
vol	is a floating-point number representing the low value of the output voltage in volts
VOH=	is the four-character keyword "VOH="
voh	is a floating-point number which defines the high value of the output voltage in volts and must be larger than *vol*
RIN=	is the four-character keyword "RIN="
rin	is a floating-point number which defines the input resistance
ROUT=	is the five-character keyword "ROUT="
rout	is a floating-point number which defines the output resistance in ohms
LOGIC=	is the six-character keyword "LOGIC="
POS	is the three-character keyword "POS"
NEG	is the three-character keyword "NEG"

The output state is equal to the result of the boolean NAND operation applied to the *k* input states.

Set-Reset Flip Flop Model

The format for the Set-Reset flip flop model statement is:

.MODEL *mname* SRFF TH=*threshold* HYSTWD=*hystwd*
+ VOL=*vol* VOH=*voh* RIN=*rin* ROUT=*rout*
+ LOGIC={POS | NEG}

where

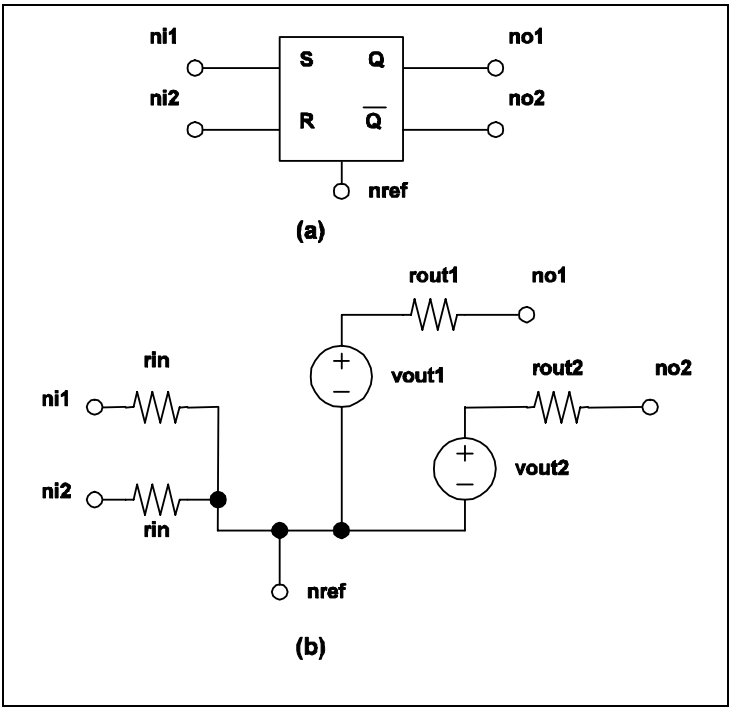
.MODEL	is the six-character keyword ".MODEL"
mname	is a legal model name as explained in "Model Names and Subcircuit Names" on page 12
SRFF	is the four-character keyword "SRFF" to stand for SRFF-type

	simple logic gates
TH=	is the three-character keyword "TH="
threshold	is a floating-point number which defines the threshold value of the input voltage in volts, which together with hystwd, determines the values of the input voltage at which the input states of the exclusive-OR gate will be changed from a logic 0 to a logic 1 and vice versa
HYSTWD=	is the seven-character keyword "HYSTWD="
hystwd	is a positive floating-point number which defines the hysteresis width of the input voltage in volts
VOL=	is the four-character keyword "VOL="
vol	is a floating-point number representing the low value of the output voltage in volts
VOH=	is the four-character keyword "VOH="
voh	is a floating-point number which defines the high value of the output voltage in volts and must be larger than the value of *vol*
RIN=	is the four-character keyword "RIN="
rin	is a floating-point number which defines the input resistance
ROUT=	is the five-character keyword "ROUT="
rout	is a floating-point number which defines the output resistance in ohms
LOGIC=	is the six-character keyword "LOGIC="
POS	is the three-character keyword "POS"
NEG	is the three-character keyword "NEG".

The actual model implemented in SIMPLIS for an S-R flip flop is shown in 4.12 (b). The set and reset input terminals of an S-R flip flop are associated with the first and second input nodes, respectively, defined in the device statement. The Q and Q' output terminals are associated with the first and second output nodes, respectively, defined in the device statement.

The logic state of the output Q' is always equal to the logical complement of the logic state of the output Q. The initial condition specified in the device statement for an S-R flip-flop is used to initialize the logic output state of the normal output Q. When the logic state of the set input is equal to logic 1, the logic state of the normal output Q is set to logic 1. When the logic state of the reset input is equal to logic 1, the logic state of the normal output Q is set to logic 0.

The output state of an S-R flip flop is supposed to be undefined when the input logic states of the set and reset inputs are both equal to logic 1. For ease of debugging, the output state of the S-R flip-flop as implemented by SIMPLIS will remain unchanged when both input states are equal to logic 1. (See 4.12)



4.12 SR Flip Flop model: (a) Symbol for a SIMPLIS S-R flip flop, (b) Model for a SIMPLIS S-R flip flop.

Clocked Set-Reset Flip-Flop

.MODEL *mname* CLK_SRFF TH=*threshold*
+ HYSTWD=*hystwd* VOL=*vol* VOH=*voh* RIN=*rin*
+ ROUT=*rout* LOGIC={POS | NEG}
+ TRIG_COND={0_TO_1 | 1_TO_0}

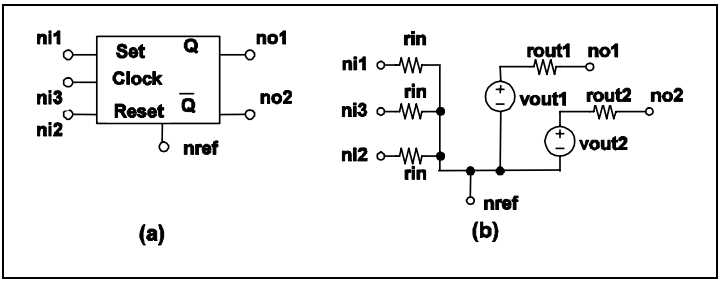
where

.MODEL	is the six-character keyword ".MODEL"
mname	is a legal model name as explained in "Model Names and Subcircuit Names" on page 12
CLK_SRFF	is the eight-character keyword "CLK_SRFF" to stand for CLK_SRFF-type simple logic gates
TH=	is the three-character keyword "TH="
threshold	is a floating-point number which defines the threshold value of the input voltage in volts, which together with hystwd,

	determines the values of the input voltage at which the input states of the exclusive-OR gate will be changed from a logic 0 to a logic 1 and vice versa
HYSTWD=	is the seven-character keyword "HYSTWD="
hystwd	is a positive floating-point number which defines the hysteresis width of the input voltage in volts
VOL=	is the four-character keyword "VOL="
vol	is a floating-point number representing the low value of the output voltage in volts
VOH=	is the four-character keyword "VOH="
voh	is a floating-point number which defines the high value of the output voltage in volts and must be larger than *vol*
RIN=	is the four-character keyword "RIN="
rin	is a floating-point number which defines the input resistance
ROUT=	is the five-character keyword "ROUT="
rout	is a floating-point number which defines the output resistance in ohms
LOGIC=	is the six-character keyword "LOGIC="
POS	is the three-character keyword "POS"
NEG	is the three-character keyword "NEG".
TRIG_COND=	is the ten-character keyword "TRIG_COND="
0_TO_1	is the six-character keyword "0_TO_1"
1_TO_0	is the six-character keyword "1_TO_0"

The actual model implemented in SIMPLIS for a clocked Set-Reset flip-flop is shown in 4.13 (b). The first two input nodes in the device statement are the set and reset input terminals while the third input node in the device statement is the clock input terminal.

If TRIG_COND = 0_TO_1, the clocked Set-Reset flip-flop is considered to be "triggered" when the logic state of the clock input changes from 0 to 1. Similarly, a logic 1 to logic 0 transition for the clock input is considered to "trigger" this type of flip-flop if TRIG_COND = 1_TO_0. The logic state of each output will not change except at the triggering moment. At the triggering moment, the logic of the clocked Set-Reset flip-flop is same as that of the unclocked Set-Reset flip-flop.



4.13 Clocked SR Flip Flop model: (a) Symbol for a SIMPLIS Clocked S-R flip flop, (b) Model for a SIMPLIS Clocked S-R flip flop.

Clocked J-K Flip-Flop

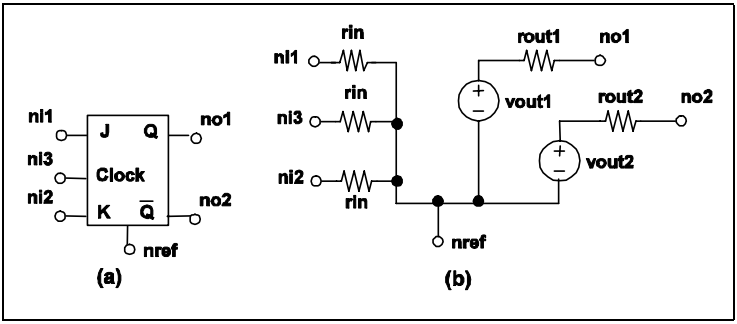
.MODEL *mname* CLK_JKFF TH=*threshold*
+ HYSTWD=*hystwd* VOL=*vol* VOH=*voh* RIN=*rin*
+ ROUT=*rout* LOGIC={POS | NEG}
+ TRIG_COND={0_TO_1 | 1_TO_0}

.MODEL	is the six-character keyword ".MODEL"
mname	is a legal model name as explained in "Model Names and Subcircuit Names" on page 12
CLK_JKFF	is the eight-character keyword "CLK_JKFF" to stand for CLK_JKFF-type simple logic gates
TH=	is the three-character keyword "TH="
threshold	is a floating-point number which defines the threshold value of the input voltage in volts, which together with hystwd, determines the values of the input voltage at which the input states of the exclusive-OR gate will be changed from a logic 0 to a logic 1 and vice versa
HYSTWD=	is the seven-character keyword "HYSTWD="
hystwd	is a positive floating-point number which defines the hysteresis width of the input voltage in volts
VOL=	is the four-character keyword "VOL="
vol	is a floating-point number representing the low value of the output voltage in volts
VOH=	is the four-character keyword "VOH="
voh	is a floating-point number which defines the high value of the output voltage in volts and must be larger than *vol*
RIN=	is the four-character keyword "RIN="
rin	is a floating-point number which defines the input resistance

ROUT=	is the five-character keyword "ROUT="
rout	is a floating-point number which defines the output resistance in ohms
LOGIC=	is the six-character keyword "LOGIC="
POS	is the three-character keyword "POS"
NEG	is the three-character keyword "NEG"
TRIG_COND=	is the ten-character keyword "TRIG_COND="
0_TO_1	is the six-character keyword "0_TO_1"
1_TO_0	is the six-character keyword "1_TO_0"

The actual model implemented in SIMPLIS for a clocked J-K flip-flop is shown in 4.14 (b). The first two input nodes in the device statement are the J and K input terminals while the third input node in the device statement is the clock input terminal.

If TRIG_COND = 0_TO_1, the clocked J-K flip-flop is considered to be "triggered" when the logic state of the clock input changes from 0 to 1. Similarly, a logic 1 to logic 0 transition for the clock input is considered to "trigger" this type of flip-flop if TRIG_COND = 1_TO_0. The logic state of each output will not change except at the triggering moment. At the triggering moment, the logic of the clocked J-K flip-flop is same as that of the unclocked Set-Reset flip-flop with one exception: if the states of both the J and the K inputs are equal to logic 1 at the triggering moment, the states of each output of a clocked J-K flip-flop will be set to the complement of its logic state right before the triggering moment. Hence, if the state of the normal output Q is equal to logic 1/0 right before the triggering moment, it will be set to logic 0/1 at the triggering moment if the states of both the J and the K inputs are equal to logic 1 at the triggering moment.



4.14 Clocked J-K Flip Flop model: (a) Symbol for a SIMPLIS Clocked J-K flip flop, (b) Model for a SIMPLIS Clocked J-K flip flop

Clocked Data Flip-Flop

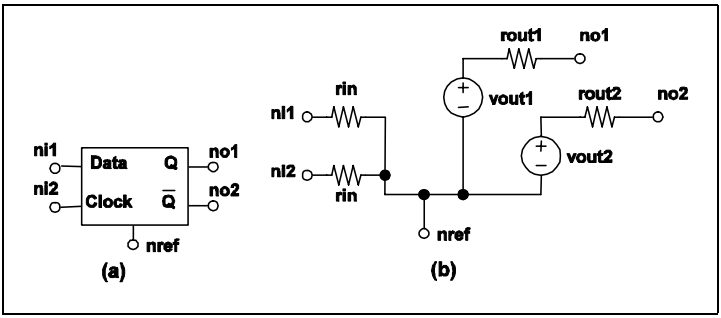
.MODEL *mname* CLK_DFF TH=*threshold* HYSTWD=*hystwd*
+ VOL=*vol* VOH=*voh* RIN=*rin* ROUT=*rout*
+ LOGIC={POS | NEG} TRIG_COND={0_TO_1 | 1_TO_0}

.MODEL	is the six-character keyword ".MODEL"
mname	is a legal model name as explained in "Model Names and Subcircuit Names" on page 12
CLK_DFF	is the seven-character keyword "CLK_DFF" to stand for CLK_DFF-type simple logic gates
TH=	is the three-character keyword "TH="
threshold	is a floating-point number which defines the threshold value of the input voltage in volts, which together with hystwd, determines the values of the input voltage at which the input states of the exclusive-OR gate will be changed from a logic 0 to a logic 1 and vice versa
HYSTWD=	is the seven-character keyword "HYSTWD="
hystwd	is a positive floating-point number which defines the hysteresis width of the input voltage in volts
VOL=	is the four-character keyword "VOL="
vol	is a floating-point number representing the low value of the output voltage in volts
VOH=	is the four-character keyword "VOH="
voh	is a floating-point number which defines the high value of the output voltage in volts and must be larger than *vol*
RIN=	is the four-character keyword "RIN="
rin	is a floating-point number which defines the input resistance
ROUT=	is the five-character keyword "ROUT="
rout	is a floating-point number which defines the output resistance in ohms
LOGIC=	is the six-character keyword "LOGIC="
POS	is the three-character keyword "POS"
NEG	is the three-character keyword "NEG"
TRIG_COND=	is the ten-character keyword "TRIG_COND="
0_TO_1	is the six-character keyword "0_TO_1"
1_TO_0	is the six-character keyword "1_TO_0"

The actual model implemented in SIMPLIS for a clocked data flip-flop is shown in 4.15 (b). The first input node in the device statement is the Data input terminal and the second input node in the device statement is the clock input terminal.

If TRIG_COND = 0_TO_1, the clocked data flip-flop is considered to be "triggered" when the logic state of the clock input changes from 0 to 1. Similarly, a logic 1 to logic 0 transition for the clock input is considered to "trigger" this type of flip-flop if TRIG_COND = 1_TO_0. The logic state of each output will not change except at the

triggering moment. At the triggering moment, the logic state of the normal output Q will follow the logic state of the data input terminal.



4.15 Clocked Data Flip Flop model: (a) Symbol for a SIMPLIS Clocked Data flip flop, (b) Model for a SIMPLIS Clocked Data flip flop

Clocked Toggle Flip-Flop

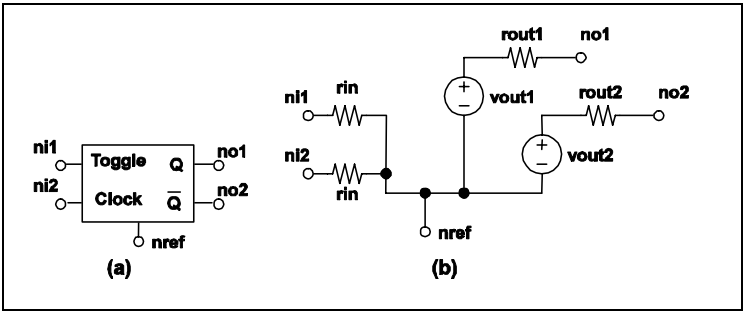
.MODEL *mname* CLK_TFF TH=*threshold* HYSTWD=*hystwd*
+ VOL=*vol* VOH=*voh* RIN=*rin* ROUT=*rout*
+ LOGIC={POS | NEG} TRIG_COND={0_TO_1 | 1_TO_0}

.MODEL	is the six-character keyword ".MODEL"
mname	is a legal model name as explained in "Model Names and Subcircuit Names" on page 12
CLK_TFF	is the seven-character keyword "CLK_TFF" to stand for CLK_TFF-type simple logic gates
TH=	is the three-character keyword "TH="
threshold	is a floating-point number which defines the threshold value of the input voltage in volts, which together with hystwd, determines the values of the input voltage at which the input states of the exclusive-OR gate will be changed from a logic 0 to a logic 1 and vice versa
HYSTWD=	is the seven-character keyword "HYSTWD="
hystwd	is a positive floating-point number which defines the hysteresis width of the input voltage in volts
VOL=	is the four-character keyword "VOL="
vol	is a floating-point number representing the low value of the output voltage in volts
VOH=	is the four-character keyword "VOH="
voh	is a floating-point number which defines the high value of the output voltage in volts and must be larger than *vol*
RIN=	is the four-character keyword "RIN="

rin	is a floating-point number which defines the input resistance
ROUT=	is the five-character keyword "ROUT="
rout	is a floating-point number which defines the output resistance in ohms
LOGIC=	is the six-character keyword "LOGIC="
POS	is the three-character keyword "POS"
NEG	is the three-character keyword "NEG"
TRIG_COND=	is the ten-character keyword "TRIG_COND="
0_TO_1	is the six-character keyword "0_TO_1"
1_TO_0	is the six-character keyword "1_TO_0"

The actual model implemented in SIMPLIS for a clocked toggle flip-flop is shown in 4.16 (b). The first input node in the device statement is the Toggle input terminal and the second input node in the device statement is the clock input terminal.

If TRIG_COND = 0_TO_1, the clocked toggle flip-flop is considered to be "triggered" when the logic state of the clock input changes from 0 to 1. Similarly, a logic 1 to logic 0 transition for the clock input is considered to "trigger" this type of flip-flop if TRIG_COND = 1_TO_0. The logic state of each output will not change except at the triggering moment. At the triggering moment, the logic state of each output remains the same as the logic state before the triggering moment if the state of the toggle input is logic 0. On the other hand, the logic state of each output is complemented if the state of the toggle input is logic 1 at the triggering moment.



4.16 Clocked Toggle Flip Flop model: (a) Symbol for a SIMPLIS Clocked Toggle flip flop, (b) Model for a SIMPLIS Clocked Toggle flip flop

Latch

.MODEL *mname* LATCH TH=*threshold* HYSTWD=*hystwd*
+ VOL=*vol* VOH=*voh* RIN=*rin* ROUT=*rout*
+ LOGIC={POS | NEG} ENABLE_LEVEL={0 | 1}

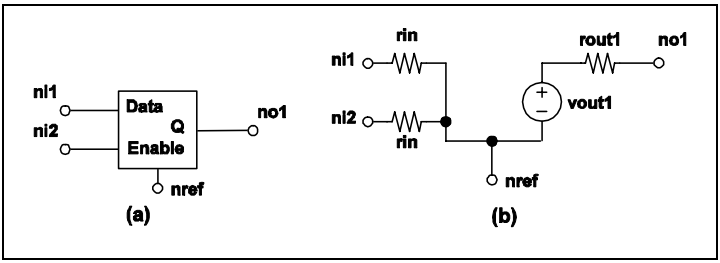
.MODEL	is the six-character keyword ".MODEL"
mname	is a legal model name as explained in "Model Names and

Subcircuit Names” on page 12

LATCH	is the five-character keyword "LATCH" to stand for LATCH-type simple logic gates
TH=	is the three-character keyword "TH="
threshold	is a floating-point number which defines the threshold value of the input voltage in volts, which together with hystwd, determines the values of the input voltage at which the input states of the exclusive-OR gate will be changed from a logic 0 to a logic 1 and vice versa
HYSTWD=	is the seven-character keyword "HYSTWD="
hystwd	is a positive floating-point number which defines the hysteresis width of the input voltage in volts
VOL=	is the four-character keyword "VOL="
vol	is a floating-point number representing the low value of the output voltage in volts
VOH=	is the four-character keyword "VOH="
voh	is a floating-point number which defines the high value of the output voltage in volts and must be larger than *vol*
RIN=	is the four-character keyword "RIN="
rin	is a floating-point number which defines the input resistance
ROUT=	is the five-character keyword "ROUT="
rout	is a floating-point number which defines the output resistance in ohms
LOGIC=	is the six-character keyword "LOGIC="
POS	is the three-character keyword "POS"
NEG	is the three-character keyword "NEG"
ENABLE_LEVEL=	is the thirteen-character keyword "ENABLE_LEVEL="

The actual model implemented in SIMPLIS for the clocked latch is shown in 4.17 (b). The first input node in the device statement is the Data input terminal and the second input node in the device statement is the enable input terminal.

If ENABLE_LEVEL = 1, the latch is considered to be "enabled" when the state of the enable input is logic 1. Similarly, if the state of the enable input is logic 0, a latch is considered to be "enabled" if ENABLE_LEVEL = 0. The logic state of the output will not change except when the latch is enabled. When the latch is enabled, the output logic state of the latch follows the logic state of the data input terminal.



4.17 Latch model: (a) Symbol for a SIMPLIS Latch, (b) Model for a SIMPLIS Latch

Chapter 5 Subcircuit Definition

Overview

The basic device elements supported by SIMPLIS -- linear resistors, linear inductors and capacitors, independent voltage and current sources, mutual inductances, four types of linear controlled sources, ideal transformers, simple switches and simple transistor switches, PWL resistors, PWL inductors and capacitors, and simple logic gates -- are very versatile and they can be used as building blocks to model a wide spectrum of electronic devices and circuits. For example, an NPN bipolar transistor can be modeled by a piecewise-linear Ebers-Moll model, by using piecewise-linear resistors for the junction diodes and two current-controlled current sources to model the current conduction. In addition, three linear resistors can be inserted to model the contact resistances as shown in 5.1

The physical transistor shown in 5.1 (a) has three nodes while the corresponding model in 5.1 (b) has six nodes, corresponding to the three terminals of the physical transistor and three internal nodes. If the physical transistor in 5.1 (a) appears only once in the entire circuit, then the model circuit in 5.1 (b) can be entered and defined in the input file as is. If the circuit contains several instances of the same device, then the model circuit in 5.1 (b) has to be repeatedly defined. For each incidence, care must be given to make sure that

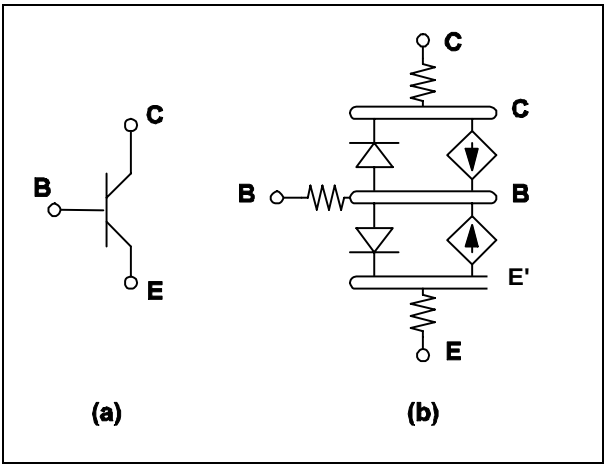
1. The node names are unique compared to the other similar definitions, and
2. Unique names are given for the seven basic elements in each definition.

Obviously, this can be quite a tedious and error prone task when the size of the circuit gets larger.

If the network shown in 5.1 (b) is defined as a subcircuit instead, every incidence of the physical transistor in the circuit can be described by the following three steps:

1. give each instance of the same type of transistor a unique device name
2. define unique node names for the three terminals of each transistor
3. reference the subcircuit defining the type of transistor involved

The tedious and error prone steps involved in giving unique names to the seven device elements in the model of 5.1 (b) and giving unique names to the three internal nodes in the model are automatically handled by the subcircuit feature of SIMPLIS. The definition and the usage of the subcircuit feature of SIMPLIS is explained in detail in this chapter.



5.1 Ebers-Moll model of an NPN transistor: (a) Circuit symbol, (b) the Ebers-Moll model. Note: B', C', and E' are the internal nodes introduced for modeling purposes

Subcircuit Definition

The main circuit refers to the circuit definition which begins with the title statement, which is the first line in the input file, and ends with the .END statement, which is the last significant line in the input file. Any number of subcircuits can be defined within the main circuit. The subcircuits can also be nested within other subcircuits. As many as 20 levels of nesting are allowed. A typical group of statements defining a subcircuit definition duplicates the following pattern of statements:

Start Subcircuit Statement
Comment Statements
Device Statements
Model Statements
Subcircuit Definitions
Control Statements
End Subcircuit Statement

Obviously, the "Start Subcircuit Statement" and the "End Subcircuit Statement" must be the first and the last statements, respectively, in the definition of a subcircuit. Between these start and end statements, the comment statements, device statements, model statements, subcircuit definitions, and control statements can appear in any order or sequence without any effect on the reading of the input file.

Parent and Child Relationships for Subcircuits

When a subcircuit named "AAA" is defined in a general circuit named "BBB", then the subcircuit "AAA" is considered the child of the general circuit "BBB" and the general circuit "BBB" is considered the parent of the subcircuit "AAA". Each subcircuit can

have only one parent and each general circuit can have zero, one, or more children. The main circuit is the ancestor of all subcircuits defined in the entire input file and it does not have a parent.

.SUBCKT Statement

In this section, a brief description of the .SUBCKT statement is given. The format of the .SUBCKT statement is defined as:

.SUBCKT *sname n1 n2 n3 ...*

where

.SUBCKT	is the seven-character keyword ".SUBCKT" signifying the start of the subcircuit definition
sname	is a legal subcircuit name as explained in "Model Names and Subcircuit Names" on page 12. A subcircuit name must be unique within a general circuit. If a name is used as a subcircuit name in a general circuit, it cannot be used as a model name in the same general circuit and vice-versa
n1	is the node name of the first external node of the subcircuit
n2	is the node name of the second external node of the subcircuit
n3	is the node name of the third external node of the subcircuit, and so on

The elements defined in a subcircuit interact with the subcircuit's parent circuit only through the subcircuit's external nodes and the ground node (node 0). Node 0, the ground node, is not allowed to be used as an external node unless the option of MAPNODE0 is used in an .OPTION control statement. Refer to Sections "Scope of Definition for a Device and for a Node" on page 90 and "Option Statements" on page 95 on the properties of MAPNODE0.

.ENDS Statement (End of Subcircuit Statement)

In this section, a brief description of the .ENDS statement is given. The format of the .ENDS statement is defined as:

.ENDS [*sname*]

where

.ENDS	is the five-character keyword ".ENDS" signifying the end of one or more subcircuits
sname	is the name of a subcircuit whose definition has not been terminated. The subcircuit name may be omitted to form the special case of the non-specific .ENDS statement.

Subcircuits are normally terminated with a .ENDS statement in the .ENDS [sname] , or specific form. If a subcircuit has not been terminated with this form of .ENDS statement, the non-specific form of the .ENDS statement:

.ENDS

will terminate the subcircuit. All subcircuits whose definition have not been terminated individually will be terminated in a group by the .ENDS statement without a subcircuit name. For example, in the statements shown in the example below, the three subcircuits SUB1, SUB2, and SUB3 are all terminated by the .ENDS statement. In this example, because of the use of the non-specific form of the .ENDS statement, subcircuit SUB1 is the parent of subcircuit SUB2, which is itself the parent of subcircuit SUB3. The placement of .ENDS statements determines the parent-child relationships of the subcircuits. The line immediately following the .ENDS statement is considered to be part of the definition of the parent of subcircuit SUB1.

```
.SUBCKT SUB113
R1 1 2 1K
C1 2 3 1U IC=1
X1 2 3 SUB2
.SUBCKT SUB2 101 103
R2 101 102 1K
C2 102 103 1U IC=1
X2 102 103 SUB3
.SUBCKT SUB3 201 203
R3 201 202 1K
C3 202 203 1U IC=1
.ENDS
```

Example 5.1

If *sname* is given in an .ENDS statement, then *sname* must be the name of the subcircuit currently defined or the name of a subcircuit which is an ancestor of the subcircuit currently defined. In this case, the definition of the current subcircuit, its parent, its grandparent, ..., and the subcircuit whose name matches *sname* are all terminated at this .ENDS statement. For the statements in the example below, the statement .ENDS SUB2 terminates the definition of subcircuits SUB2 and SUB3 but not the definition of subcircuit SUB1. The line immediately following the ".ENDS SUB2" statement is considered part of the definition of subcircuit SUB1, which is the parent of subcircuit SUB2.

```
.SUBCKT SUB113
R1 1 2 1K
C1 2 3 1U IC=1
X1 2 3 SUB2
.SUBCKT SUB2 101 103
R2 101 102 1K
C2 102 103 1U IC=1
X2 102 103 SUB3
.SUBCKT SUB3 201 203
R3 201 202 1K
C3 202 203 1U IC=1
.ENDS SUB2
```

Example 5.2

Notice that there is a unique correspondence between each .SUBCKT statement and the start of a subcircuit while each .ENDS statement may be "shared" by several subcircuits. The recommended practice, however, is to terminate each subcircuit with a unique .ENDS statement having a matching subcircuit name instead of using the

implied termination. This will make reading the input file easier and will make the subcircuit definitions less susceptible to error.

Scope of Definition

As pointed out in “Sequence of Statements” on page 17, the scope of definition for a "general circuit" begins at the Start Circuit Statement and stops at the End Circuit Statement, inclusively. The concept of the scope of definition is formally defined in this section.

The Scope of Definition for the Main Circuit

The scope of definition of the main circuit, which is the highest level of any circuit specified in the input file, ranges from the title statement to the .END statement, inclusively.

The Scope of Definition for a Subcircuit

The scope of definition for a subcircuit ranges from its start at the subcircuit statement to the .ENDS statement that terminates it, inclusively. The user is reminded that it is possible for several subcircuits to share the same .ENDS statement to terminate their definition. The scope of definition of a child subcircuit must be inside the scope of definition of its parent circuit.

Scope of Definition for a Device and for a Node

The scope of definition for a device is the "youngest" subcircuit whose scope of definition encompasses the device statement of the device in question. For the statements shown in the example below, the capacitor CB is considered to be defined in the subcircuit "SUB2" whereas the capacitor CA is considered to be defined in the subcircuit "SUB1".

Each device is considered to be local in its subcircuit in the sense that its name is only "made known" to this subcircuit and its name is not made available to other subcircuits. In the example below, the device names CA, RA, and XSUB are known only within the subcircuit SUB1, but not in the subcircuit SUB2. Similarly, the device names CB and RB are known only within the subcircuit SUB2. For devices which are controlled by the voltage or current of another device, such as some controlled sources, simple switches, and simple transistor switches, the controlling device must also be defined in the same circuit as the controlled device or SIMPLIS will not be able to locate the controlling device.

Similarly, the scope of definition for a node is the "youngest" subcircuit whose scope of definition encompasses the node name in question. For the statements in the example below, the nodes 101, 102, and 103 are considered to be defined in the subcircuit "SUB2" whereas the nodes 1, 2, and 3 are considered to be defined in the subcircuit "SUB1". Each node is considered to be local in its circuit and its name is not made available to other circuits. This rule for the scope of definition of a node applies to all nodes defined in a circuit except for node 0 .

If the option MAPNODE0 is turned off, which is the default case, node 0 in any subcircuit is treated as the same node as node 0 in the main circuit. If the option

MAPNODE0 is turned on through an .OPTION statement, then node 0 in any subcircuit is considered to be defined locally in that particular subcircuit.

Since each device or node is only locally defined in its circuit of definition, it is acceptable to have the same device names and the node names used in different circuits.

```
.SUBCKT SUB1 1 3
CA 1 2 10U IC=2
RA 2 3 10K
XSUB 2 3 SUB2
.SUBCKT SUB2 101 103
CB 101 102 10U IC=2
RB 102 103 10K
...
...
.ENDS SUB2
.ENDS SUB1
```

Example 5.3

External and Local Nodes

Nodes defined in the .SUBCKT statement are called the external nodes of the subcircuit. Hence, node 1 in the subcircuit "SUB1" in the example below is an external node and node 101 is an external node for subcircuit "SUB2". The statements in the example below represent the complete input file of an example system to be studied. The .TRAN statement in this input file has not yet been covered but its presence does not affect our discussion here.

In the definition of the subcircuit "SUB1" in the example below, nodes 1, 2, 3, and 0 appear in the device statements. Unless the MAPNODE0 option is turned on through an .OPTION statement, node 0 in a subcircuit is considered to be global in the sense that it is treated to be the same node as node 0 in the main circuit. Any node in the device statements of a subcircuit that is neither an external node nor a global node is considered a local node. So in this example, nodes 2 and 3 are local nodes in subcircuit "SUB1" and node 102 is a local node for subcircuit "SUB2".

```
This is the title line

V1                1    0    DC   10
R1                1    2    1K
R2                2    3    1K
R3                3    0    1K
X1                2    SUB1

.SUBCKT                SUB1 1
R1                1    2    1K
C1                2    0    1U
R2                2    3    10K
C2                3    0    1U   IC=0
XA                2    SUB2
XB                3    SUB2

.SUBCKT                SUB2 101
R101              101  102  1K
C101              102  0    1U   IC=0
.ENDS SUB2

.ENDS SUB1
...
...
.TRAN 1 0
.END
```

Example 5.4

Subcircuit Calls/Instantiation

The format for a statement defining a subcircuit call or instantiation has been elaborated in and is repeated here for convenience:

Xname n1 n2 ... sname

Such a device statement has a device name that begins with the one-character keyword, "X", followed by a set of node names, and the name of a subcircuit at the end.

In order for SIMPLIS to be able to find the referenced subcircuit, the subcircuit instantiation and the subcircuit named "*sname*" must be defined in the same general circuit. In addition, the number of nodes listed in the subcircuit instantiation must match the number of external nodes listed in the subcircuit definition.

During the reading of the input file, SIMPLIS first combs through the main circuit to register the node names that have already been employed in the main circuit. In example 5.3, the nodes 0, 1, 2, and 3 have been utilized in the main circuit. These node names form a list of existing nodes so that new node names introduced later during subcircuit instantiation will not duplicate these existing node names. Similarly, a list of device names already employed in the main circuit is created. For this example, this list includes V1, R1, R2, and R3. Sometimes devices such as the one represented by X1 are referred to as "pseudo devices" because there is no actual device element corresponding to the keyword "X."

After reading the main circuit, SIMPLIS reads the circuit instantiation statements in the main circuit and starts instantiating the subcircuits. Each subcircuit instantiation can be summarized in a four-step procedure:

1. Perform a one-to-one mapping of the names of external nodes to the names of corresponding nodes in the subcircuit instantiation statement.
2. Map the name of each local node to a new node name, different from any existing node name. This new node name is then added to the list of existing node names.
3. Map the name of each device defined in the subcircuit to a new device name having the same element keyword, but an individual name different from any existing one. This new device name is then added to the list of existing device names.
4. Carry out subcircuit instantiation for each subcircuit instantiation statement in the current subcircuit.

In example 5.3, node 1 of subcircuit "SUB1" would be mapped to node 2 when the subcircuit instantiation for the pseudo-device X1 is carried out. Then local nodes 2 and 3 in "SUB1" are each mapped to a new node name. The device names R1 and R2 are each mapped to a new name for a linear resistor and device names C1 and C2 are each mapped to a new name for a linear capacitor. The actual new node names or device names introduced in the mapping are not important to the user as they are only used internally by SIMPLIS. For the purpose of illustration here, let us assume that nodes 2 and 3 in "SUB1" are mapped to nodes 7 and 9, respectively.

When SIMPLIS carries out the subcircuit instantiation for the pseudo-devices XA in the subcircuit definition of "SUB1", the four-step procedure is repeated:

1. Node 101 of subcircuit "SUB2" is mapped to node 2 in subcircuit "SUB1". Since node 2 in "SUB1" has already been mapped to node 7, node 101 of "SUB2" is mapped to node 7.
2. Node 102, the only local node of subcircuit "SUB2" is mapped to a new node name that is not in the list of existing nodes. Then this new node name is added to the list of existing nodes.
3. Devices R101 and C101 are mapped to appropriate new device names that are different from existing device names. Then these device names are added to the list of existing device names.
4. Since there is no subcircuit defined in the subcircuit "SUB2", the instantiation of XA has been completed.

When SIMPLIS carries out the subcircuit instantiation for the pseudo-devices XB in the subcircuit definition of "SUB1", the four-step procedure is repeated:

1. Node 101 of subcircuit "SUB2" is mapped to node 3 in subcircuit "SUB1". Since node 3 in "SUB1" has already been mapped to node 9, node 101 of "SUB2" is mapped to node 9.
2. Node 102 is mapped to a new node name that is not in the list of existing nodes. Then this new node name is added to the list of existing nodes.
3. Devices R101 and C101 are mapped to appropriate new device names that are different from existing device names. Then these device names are added to the list of existing device names.

4. Since there is no subcircuit defined in the subcircuit `SUB2", the instantiation of XB has been completed.

The four-step procedure is repeated until all subcircuits have been instantiated. Notice that the definition of a subcircuit only provides a model subcircuit with a reference name. The subcircuit does not come into being until the appropriate subcircuit instantiation.

Chapter 6 Control Statements

Overview

The device statements, model statements, and subcircuit definition statements discussed in Chapters through See are all related to circuit definition. There are additional statements that control the type of analysis to be performed and the type of output data to be generated by SIMPLIS. These types of statements are not related to circuit definition. They are collectively called the control statements.

All control statements start with the period (`.') as the first character in the statement. However, not all statements starting with a period are control statements. The exceptions are the .MODEL, .END, .SUBCKT and .ENDS statements. A control statement can be classified as one of the following types:

1. Options
2. Initial conditions
3. Printing of variables
4. Analyses

All types of control statements can appear within the scope of definition of the main circuit but only the control statements related to the setting of initial conditions and the printing of variables are allowed to appear inside the scope of definition of a subcircuit. In general, control statements can appear in any order of sequence. The one exception to this rule is the order of the analysis statement, which dictates the sequence in which SIMPLIS performs the different analyses.

Option Statements

As its name implies, option statements are used to set various options to appropriate values. More than one option statement can appear in the input file. The format of an option statement is:

.OPTIONS *opt1 opt2* ...

where *opt1*, *opt2*, and ..., are various options. Some options only take on the values of ON or OFF. The defaults values for such options are OFF and the option values are turned ON by having the corresponding keyword present in an option statement. Some other options assume the form of the parameter assignment outlined in . Option statements are not allowed to be placed within the scope of definition of any subcircuit because the options apply to the entire system under study. The various options and their meanings are:

EMSG_MAX=*k* Sets the Maximum number of error messages that are generated before the syntax checking of the input file is suspended. "EMSG_MAX=" is the nine-character keyword "EMSG_MAX=" and *k* is a positive integer. For very severe syntax errors, the syntax checking is suspended before the number of error messages reaches this maximum limit. The default value for *k* is 20.

EXPAND	Show the entire circuit after all subcircuit calls have been instantiated. The listing of this expanded circuit is shown in the file "XXXX.lst" where XXXX is the name of the input file. The default is not to show the expanded circuit.
MAPNODE0	By default, node 0 is not allowed to appear as an external node in the definition of a subcircuit and node 0 in a subcircuit is considered the same node as node 0 in the main circuit. If MAPNODE0 is specified as an option, node 0 in a subcircuit would not be considered as the same node as node 0 in the main circuit. In such a case, node 0 in the subcircuit is appropriately mapped for each subcircuit instantiation and it is allowed to appear as an external node in the subcircuit definition.
PSP_START=t1	For time-domain transient analysis, the print variables are generated for time values larger than or equal to the value of t1. "PSP_START=" is the ten-character keyword "PSP_START=" and t1 is a nonnegative floating-point number. If this option is not specified, the default value for t1 is the time the simulation starts saving data for the transient analysis. Refer to the .TRAN statement for details of the transient analysis.
PSP_END=t2	For time-domain transient analysis, the print variables are generated for time values up to but not larger than the value of t2. "PSP_END=" is the eight-character keyword "PSP_END=" and t2 is a positive number larger than the value of t1. If this option is not specified, the default value for t2 is the stop time of the transient analysis as specified in the .TRAN statement. If the value of t2 in PSP_END is smaller than the value for t1 in PSP_START, no output print file will be generated for the transient analysis.
PSP_NPT=n	Set the minimum number of data points generated for printing the output variables during the transient analysis. "PSP_NPT=" is an eight-character keyword and n is an integer between 2 and 8001, inclusively. (PSP_END - PSP_START)/(PSP_NPT - 1) is the step size of the time variable in the print file generated for the transient analysis. If PSP_NPT is not specified, no output print file will be generated for the transient analysis.
POP_ITRMAX=n	POP Iteration Limit. This option sets the maximum number of iterations for the POP analysis. If set, the option value must be a positive integer between 1 and 200, inclusively. If this option is not set, it defaults to 20. Example: .OPTIONS POP_ITRMAX=50
POP_USE_TRAN_SNAPSHOT	This option instructs POP to take advantage of the last data point of a previous transient simulation, assuming the circuit

and the initial conditions remained the same between the two simulation runs.

This option does not take on any value. It is either turned ON or turned OFF. If it is turned ON, the POP analysis will use the last data point of a previous transient simulation as the initial condition to start the POP analysis. Usually this leads to a faster POP analysis.

Example:

.OPTIONS POP_USE_TRAN_SNAPSHOT

POP_OUTPUT_CYCLES=n

Number of cycles of steady-state POP Data to show. After a successful POP analysis, SIMPLIS will generate the steady-state time-domain waveforms for an integral number switching cycles.

If set, the option value must be a positive integer between 1 and 16, inclusively.

If this option is not set, it defaults to 5.

Example:

.OPTIONS POP_OUTPUT_CYCLES=3

POP_SHOWDATA

Display POP Data. In general, this option is turned on as a debugging aid if a Periodic Operating Point (POP) analysis fails.

This option does not take on any value. It is either turned ON or turned OFF. If it is turned ON, the progress of the periodic operating point analysis is output and the resulting waveforms may be viewed in the same manner as for the POP cycles. To turn this option ON this, the line in the following example should be output to the SIMPLIS simulation input deck.

Example:

.OPTIONS POP_SHOWDATA

MAX_TOPOLOGY Maximum number of saved topologies. This sets the maximum number of topologies that SIMPLIS would save during a simulation run. Topologies are saved in a cache for possible reuse to speed up the run. This option sets the size of the cache.

If set, the option value must be a positive integer between 2 and 65536, inclusively.

If this option is not set, it defaults to 1024.

Example:

OPTIONS MAX_TOPOLOGY=512

SNAPSHOT_INTVL Minimum duration between snapshots. This instructs SIMPLIS to save a snapshot of the internal state of the simulation at intervals no smaller than the option value set for SNAPSHOT_INTVL.

If set, the option value must be a non-negative floating-point number. Engineering prefixes are allowed.

If this option is not set, it defaults to zero. If this option is not set, or if the option is set to zero, SIMPLIS would not save any snapshots at all.

Example:

.OPTIONS SNAPSHOT_INTVL =10M

SNAPSHOT_NPT Maximum number of saved snapshots. This sets the maximum number of snapshots that SIMPLIS would save. If there is a conflict between the values set for the SNAPSHOT_NPT and SNAPSHOT_INTVL options, the value set for the SNAPSHOT_NPT option will override the value set for the SNAPSHOT_INTVL option.

If set, the option value must be a non-negative number between 11 and 201, inclusively.

If this option is not set, SIMPLIS will not save any snapshot.

Example:
.OPTIONS "SNAPSHOT_NPT=30

NEW_ANALYSIS Force new analysis. This option does not take on any value. It is either turned ON or turned OFF. If it is turned ON, SIMPLIS will ignore any relevant data files from a previous simulation, even if the circuit and the initial conditions between the two simulation runs are the same. If this option is turned OFF, SIMPLIS will try to take advantage of any relevant data files from the previous simulation if the circuit and the initial conditions have not changed from the previous simulation run.

To turn ON this option, the line in the following “Example” section should be output to the SIMPLIS simulation input deck.

.OPTIONS NEW_ANALYSIS

FREQ_DOMAIN Domain in small-signal AC analysis. The small-signal AC analysis can be carried out in the continuous (s-) domain or in the discrete (z-) domain.

If set, the option value must be either the character ‘S’ or ‘Z.’

If this option is not set, it defaults to ‘S.’

Example:

.OPTIONS FREQ_DOMAIN = Z

IGNORE_UNITS — Ignore Units. This option does not take on any value. It is either turned ON or turned OFF. If it is turned ON, SIMPLIS will ignore any trailing units or strings when it is looking for a floating-point number. For example, in specifying the voltage of a DC voltage source, SIMPLIS would not complain the trailing 'V' here 1.23V if this options is turned ON.

The default is to not to turn ON this option and SIMPLIS would then complain about the trailing string or units. To turn ON this option, the line in the following "Example" section should be output to the SIMPLIS simulation input deck.

Example:

.OPTIONS IGNORE_UNITS

Control Statements for Setting Initial Conditions

As outlined in , initial conditions are required to be supplied in the device statements. However, an .INIT statement can be used to override the initial conditions supplied in the device statements. .INIT statements are allowed to be placed within the scope of definition of a subcircuit and more than one .INIT statement can appear within the same scope of definition.

Linear and PWL Capacitors

The .INIT statement can be used to override the initial voltage of a linear capacitor or a PWL capacitor. For example,

```
.INIT V(C11)=0 V(!C23)=12.0
```

means that the initial voltage of linear capacitor C11 is set to 0 V while the initial voltage of PWL capacitor !C23 is set to 12 V. C11 and !C23 must be in the same scope of definition as the .INIT statement. Notice that more than one initial condition setting can be supplied in the same .INIT statement. Each initial condition setting is in the form of a parameter assignment as outlined in . Whatever initial voltages were specified for C11 and !C23 in the device statements, they are superseded by 0 V and 12 V, respectively.

Linear and PWL Inductors

The .INIT statement can be used to override the initial current of a linear inductor or a PWL inductor. For example,

```
.INIT I(L12)=0 I(!L22)= -2m
```

means that the initial current of linear inductor L12 is set to 0 A whereas the initial current of PWL inductor !L22 is set to -2 milliamperes.

Setting of Initial States for S and Q Switches

The .INIT statement can be used to override the initial states of simple switches and simple transistor switches:

```
.INIT Q1=OPEN SA=CLOSE
```

means that the initial state of the Q switch Q1 is set to OPEN whereas the initial state of the S switch SA is set to CLOSE.

Setting of Initial Segment for PWL Resistors

The .INIT statement can be used to override the initial segment of operation for PWL resistors:

```
.INIT !R100=3
```

means that the initial segment of operation for the PWL resistor !R100 is set to the 3rd segment of this device.

Setting of Initial State for Simple Logic Gates

The .INIT statement can be used to override the initial state of a simple logic gate. For example,

```
.INIT !D9=0 !D8=1
```

means that the initial output state for logic gate !D9 is set to logic 0 whereas the initial output state for logic gate !D8 is set to logic 1.

Initial Conditions for Devices in a Subcircuit

Since initial conditions are either specified in the device statements or through the .INIT statements discussed so far, two subcircuit instantiations referencing the same subcircuit definition naturally have identical initial conditions. Let us examine the statements in example 6.1 (a). The capacitor, originally named CA in subcircuit "SUB1", has an initial branch voltage of 1 V in the subcircuit instantiation of both X1 and X2.

In some situation, it is desirable to be able to set the initial condition of a device in a subcircuit to different values for different subcircuit instantiations. SIMPLIS allows the initial conditions for devices in a child subcircuit be overridden with an .INIT statement in the parent circuit. The extra .INIT statement shown in example 6.1 (b) means that the initial voltage on capacitor CA in subcircuit "SUB1" for the instantiations X1 and X2 should be 5 V and 2 V, respectively.

The capability to override device initial conditions can be extended to lower levels of subcircuits. For example, the expression

```
.INIT X123.X456.!D9=1
```

means that the logic gate !D9 in the subcircuit referred to as X456 in a subcircuit referred to as X123 in the current circuit is set to have an initial output state of logic 1.

The .INIT statement in an ancestor circuit always overrides any initial condition specified in a subcircuit. For example, if

```
.INIT I(L12)=0
```

appears within the scope of definition of a subcircuit referred to as X123 in the main circuit and

```
.INIT I(X123.L12)=1
```

appears within the scope of definition for the main circuit, the initial current for this inductor is set to 1 A, not 0 A.

```
X1 1 2 SUB1
X2 9 8 SUB1
.SUBCKT SUB1 101 102
RA 101 103 1K
CA 103 102 1U IC=1
.ENDS SUB1
```

Example 6.1a

```
X1 1 2 SUB1
X2 9 8 SUB1
.SUBCKT SUB1 101 102
RA 101 103 1K
CA 103 102 1U IC=1
.ENDS SUB1
.INIT V(X2.CA)=2 V(X1.CA)=5
```

Example 6.1b

Control Statements for Printing Variables

The .PRINT statement is used to specify the output variables to be recorded for printing/plotting. Note that with the SIMetrix/SIMPLIS, .PRINT only specifies data to be saved in the binary file and does not create ASCII tabular output. .PRINT instructs SIMPLIS to send the specified data to the SIMetrix front end. SIMetrix saves the data as a vector in its binary file. The actual vector name used is described in the following paragraphs.

The format of .PRINT is:

.PRINT *var1 var2 ...*

where *var1*, *var2*, and ... are legal print variables. Forms of legal print variables and their meanings are listed below:

V(*DName*) Branch voltage across a two-terminal device in the current circuit. DName is the device name of a device whose element keyword is one of the following:

R, L, C, V, I, E, G, H, F, Q, S, !R, !L, and !C

SIMetrix vector name will be *DName*

V(*#NodeName)* Voltage on mapped node *NodeName*. Mapped nodes are created using the .NODE_MAP statement.

SIMetrix vector name will be *#NodeName*.

I(*DName*) Branch current through a two-terminal device in the current circuit. DName is the device name of a device whose element keyword is one of the following:

R, L, C, V, I, E, G, H, F, Q, S, !R, !L, and !C

SIMetrix vector name will be of the form:

DName#pinname

where *pinname* will be *p* for the first pin and *n* for the second pin

I(*DName#pinname*) Current through a device pin. *DName* is the device name while *pinname* is either a pin number or mapped pin name. If *Dname* is a subcircuit device and the subcircuit definition includes .NODE_MAP statements to map its external nodes to names, them those mapped names may be used for *pinname*. For example:

```
X1 1 2 3 SUB1
.SUBCKT SUB1 100 200 300
.NODE_MAP INP 100
.NODE_MAP INN 200
.NODE_MAP OUT 300
...
...
.ENDS SUB1

.PRINT I(X1#INP)
```

The above .PRINT will instruct SIMPLIS to output the current into pin connected to node 1 of X1. The resulting SIMetrix vector will be called "X1#INP".

V(*Node1*,*Node2*) Differential voltage from node *Node1* to node *Node2* where *Node1* and *Node2* are node names in the current circuit

V(*Node1*) Voltage from *Node1* to node 0, the ground node. This form is allowable only in the main circuit and only if node 0 is present in the main circuit.

SIMetrix vector name will be:

Node1

V(*Xname1*.*Xname2*.*DName*)
Branch voltage across the device named *DName* in the subcircuit referred to as *Xname2* in the subcircuit referred to as *Xname1* in the current circuit. The allowable device is same as

those listed for the form of V(*DName*).

SIMetrix vector name will be:

Xname1.Xname2.DName

I(*Xname1.Xname2.DName*)
Branch current through the device named *DName* in the subcircuit referred to as *Xname2* in the subcircuit referred to as *Xname1* in the current circuit.

SIMetrix vector name will be of the form:

Xname1.Xname2.DName#pinname

where *pinname* will be *p* for the first pin and *n* for the second pin

V(*Xname1.Xname2.Node1,Node2*)
Differential voltage from node *Node1* to node *Node2* where *Node1* and *Node2* are node numbers in the subcircuit referred to as *Xname2* in the subcircuit referred to as *Xname1* in the current circuit.

NODE_V — Print all node voltages in the main circuit with respect to the ground node, node 0, in the main circuit.

ALL — Print all node voltages in the main circuit with respect to the ground node, node 0, in the main circuit and print all branch currents of the two-terminal devices in the main circuit. The two-terminal devices refer to those whose branch current can be printed through the form of I(DName).

In the normal case, no more than 50 output variables can be created for printing/plotting in one simulation. If more than 50 output variables are needed, the output variables can be generated through more than one pass of the simulation.

If the print variable NODE_V or ALL is used, then node 0 must be present in the main circuit. Although the number of output variables that are associated with NODE_V or ALL are obviously large, NODE_V or ALL is counted as one print variable in counting towards the maximum number of 50 print variables allowed in one simulation. Simulation can be substantially slower with NODE_V or ALL as a printing variable. In addition, if ALL is used as the print variable, other print variables defined in the main circuit, or in any of its descendant subcircuits, are ignored.

Mapping Names to Node Numbers

The SIMPLIS input deck format requires all nodes to be defined as numbers. The names used for the SIMetrix vectors for the voltage on a node always use the name of that node. So the SIMetrix vector names will be numbers as well. This can be inconvenient, so a method is available to instruct SIMPLIS to use a user defined name for any node voltage vector. This is done using the .NODE_MAP statement. The format of the .NODE_MAP statement is:

.NODE_MAP *mapped_name node_number*

where:

mapped_name	User defined name. Must have at lease one alphabetic character and may comprise any alphanumeric character in addition to the underscore ('_').
node_number	Node number to be mapped.

For example

```
.NODE_MAP VOUT 27
```

If the above line is included in the SIMPLIS input deck, the vector for the voltage at node 27 will be named "VOUT".

The .NODE_MAP statement may also be used inside subcircuits. If such a .NODE_MAP statement is used to map an external node, then .NODE_MAP statements must be issued for all external nodes. Mapping external nodes also controls the naming of current vectors into the subcircuit's terminals. See “Control Statements for Printing Variables” on page 101 for further details.

Creating SIMetrix Plots

SIMPLIS supports the .GRAPH statement for creating plots while the simulation proceeds. For full documentation on .GRAPH, please refer to the *SIMetrix Simulator Reference Manual*. Note that the names used for signals in .GRAPH must comply with SIMetrix vector names rather than the format SIMPLIS uses for .PRINT. For example, the following instructs SIMPLIS to save and plot the voltage on node 2:

```
.PRINT V(2)
.GRAPH :2 CurveLabel="Voltage at Node2"
```

Note that '2' is prefixed with a colon. This is to distinguish node '2' from the constant value 2.0.

Another example:

```
.NODE_MAP VOUT 224
.PRINT V(#VOUT)
.GRAPH #VOUT CurveLabel="Output voltage"
```

In the above, node 224 has been mapped to the name "VOUT". Subsequently, the data on node 224 may be referenced as #VOUT.

Control Statements Associated with Analyses

The analyses supported by SIMPLIS at this point are the time-domain transient analysis, the periodic operating point analysis, and the frequency-domain small-signal (AC) analysis.

.TRAN - Time-Domain Transient Analysis

The .TRAN statement instructs SIMPLIS to perform a time-domain transient analysis. The initial conditions for various devices are taken from the device statements and any

overriding initial conditions supplied from the .INIT statements. The time-domain transient analysis always starts with the time variable set to zero. The format for the .TRAN statement is

.TRAN *tstop tsave*

where

.TRAN is the five-character keyword ".TRAN".

tstop is a positive floating-point number in seconds to stand for the time instant at which the time-domain transient analysis stops

tsave is a positive floating-point number smaller than tstop.

The transient analysis starts with the time variable t equal to 0.0 and stops at t equal to tstop. The result of the simulation is not saved, however, until the time variable t reaches tsave. Output data for the time-domain transient analysis can be generated for printing or plotting for any time instant between tsave and tstop, inclusively.

.POP - Periodic Operating Point Analysis

See “Simplis-POP” on page 135 for a detailed description

.AC - Frequency Domain Analysis

See “Simplis-FX” on page 155 for a detailed description

Chapter 7 Running SIMPLIS

Overview

The version of SIMPLIS covered by this manual may only be used within the SIMetrix environment.

Usually, circuits are entered using the SIMetrix schematic editor which takes care of many of the syntax details covered elsewhere in this manual. Using SIMPLIS in this manner is covered in the *SIMetrix User's Manual* but brief details are repeated here.

It is also possible to run SIMPLIS using a netlist prepared by hand or with another schematic entry tool. This must be done within the SIMetrix environment and the following sections describe how.

Some features of the SIMetrix schematic editor that are useful for running SIMPLIS are also repeated here for convenience.

Running SIMPLIS on a SIMetrix Schematic

You must first enter the schematic in ‘SIMPLIS mode’. To select SIMPLIS mode, from the schematic window select menu File|Select Simulator then select SIMPLIS. If you have already added some components in SIMetrix mode, you may get an incompatibility warning. This means that some of the components placed will not work with SIMPLIS. These will need to be either replaced with suitable alternatives or re-entered.

When the schematic has been entered, select your chosen analysis mode using Simulator|Choose Analysis... .

You can now run the simulation by pressing F9 or menu Simulator|Run.

Adding Extra Netlist Lines

The analysis mode selected using the schematic editor's Simulator|Choose Analysis... menu is stored in text form in the schematic's *simulator command window*. If you wish, it is possible to edit this directly. Note that the text entered in the simulator command window and the Simulator|Choose Analysis... dialog settings remain synchronized so you can freely switch between the two methods.

To open the simulator command window, select the schematic then press the F11 key. It has a toggle action, pressing it again will hide it. If you have already selected an analysis mode using the Choose Analysis dialog box, you will see the simulator controls already present.

The window has a popup menu selected with the right key. The top item "Edit file at cursor" will open a text editor with the file name pointed to by the cursor or selected text item if there is one.

The simulator command window can be resized using the splitter bar between it and the schematic drawing area.

You can add anything you like to this window not just simulator commands. The contents are simply appended to the netlist before being presented to the simulator. So, you can place device models, mutual inductor specifications, .OPTION controls or simply comments. The Choose Analysis dialog will parse and possibly modify analysis controls and some .OPTIONS settings but will leave everything else intact.

Some schematics may be simulated with both SIMPLIS and the SPICE based SIMetrix simulator. The commands (referred to as 'controls' in SIMetrix documentation) for these simulators are, however, incompatible. For this reason, SIMetrix and SIMPLIS commands are separated using the .SIMULATOR control. The syntax for this is:

.SIMULATOR SIMPLIS | SIMetrix | DEFAULT

SIMPLIS	All lines following and until the next .SIMULATOR control will only be passed to the netlist in SIMPLIS mode.
SIMetrix	All lines following and until the next .SIMULATOR control will only be passed to the netlist in SIMetrix mode.
DEFAULT	All lines following and until the next .SIMULATOR control will be passed to the netlist in both modes.

Running SIMPLIS for an External Netlist

You may wish to run SIMPLIS on a netlist (also known as an 'Input Deck') created by hand or by another program.

To run a netlist from the SIMetrix GUI, select the command shell menu SIMPLIS|Run Netlist... then select the file you wish to run. Note that this will pass the netlist through the netlist pre-processor before it is presented to SIMPLIS. The pre-processor provides some additional features to import and parameterize device models. See "Netlist Preprocessor" on page 109 for details. Note that a complete syntactically correct SIMPLIS netlist will not be functionally altered by the pre-processor.

Running SIMPLIS from a Script

SIMPLIS may be launched from a script using the command RunSIMPLIS:

RunSIMPLIS *filename*

filename	Name of file containing the SIMPLIS netlist. If a full path is not supplied, *filename* will be assumed to be relative to the current directory. Note that the extension of the file must always be supplied; no default is assumed.

The RunSIMPLIS command will not pre-process the netlist. This must be done separately using the PreProcessNetlist command. See "Netlist Preprocessor" on page 109.

RunSIMPLIS is the primitive SIMetrix command that launches SIMPLIS. However, when running a simulation on a schematic, a number of other activities are performed.

These include pre-processing the netlist generated by the schematic editor and also resolving a trigger device for POP analysis. (See “Simplis-POP” on page 135) If you wish to simulate a schematic in exactly the same manner as the Run menu, you need to execute the script simplis_run. This simulates the currently open schematic. The full source for simplis_run can be found on the install CD.

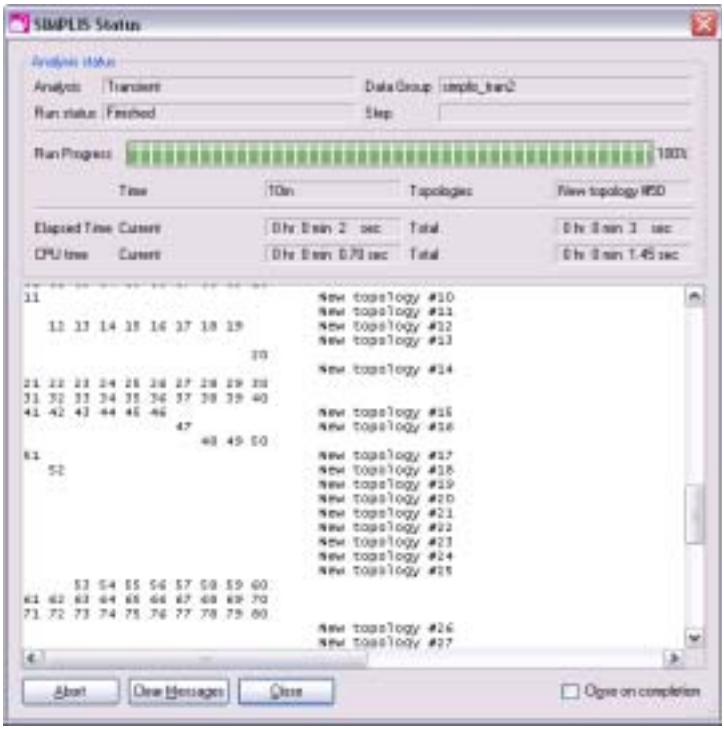
Running SIMPLIS from a DOS Prompt

The version of SIMPLIS supplied with SIMetrix/SIMPLIS cannot be run directly from the command (DOS) prompt.

SIMPLIS Execution

SIMPLIS runs as a separate process but communicates with SIMetrix while it is running to provide information on the progress of the run.

This progress is displayed in the SIMPLIS Status Window as shown below.



The message window shows some detail about the progress and is similar to that displayed in earlier versions without the status window display.

Aborting a SIMPLIS Run

At any point during a simulation run, you can abort the simulation. Select the command shell menu SIMPLIS|Abort Run or simply press the Abort button on the SIMPLIS Status Window. This will abort SIMPLIS at the next suitable point in its execution which may not be immediately. Note that there is no resume facility with SIMPLIS.

Automatic Program Suspension by SIMPLIS

Sometimes the input circuit may contain a switching conflict which SIMPLIS cannot resolve. For example, connecting the input and output nodes of a SIMPLIS inverter together creates a situation where the inverter cannot locate a valid logic output state from which to operate. Once such a switching conflict occurs, the simulation enters an endless loop and it is unable to advance forward in time. After attempting 200 times without any success in locating a correct operating state, SIMPLIS prints a message to the message window similar to the following, and aborts the simulation:

```
Unable to find a starting operating point!!
```

or

```
At t = 1.2345e-06, it is unable to find the correct state of
operation for some device.
For example, check the following devices:
!R1, !R2, and !R3
```

If such a message appears, you should carefully inspect the input file and appropriate data files generated by SIMPLIS to locate the source of the switching conflict.
In a slightly different scenario, the situation may occur when the switching logic is properly defined but a few circuit elements are connected in a manner which may cause the simulation to go through very fast and repetitive switching in a small group of states. If the time interval between two switching events is so short to be negligible compared to the actual simulation time variable, SIMPLIS aborts the simulation and displays a similar message to the following:

```
No advance in the time variable
at t = 3.1425e-05 sec. !!
```

This example error message means that when the time variable, t, reaches 3.1425x10 -5 sec in the simulation, two consecutive switching events have occurred within a time span that is negligible compared to 3.1425x10 -5 sec. A number is considered to be negligible to another number if it is sixteen orders of magnitude below the larger number. Again, you should then inspect the input file and appropriate data files generated by SIMPLIS to locate the source of the problem.

Netlist Preprocessor

Overview

The netlist preprocessor is actually part of SIMetrix not SIMPLIS. However, although it can be used with SIMetrix (SPICE) netlists, it was originally developed for use with SIMPLIS and so is documented here. The netlist preprocessor provides some additional functionality not provided by the simulator itself. These functions are:

1. Searches the model library for unresolved subcircuits and adds them to the netlist.
2. Evaluates parameterised expressions.
3. Builds static subcircuits from parameterised definitions
4. Localizes globally defined subcircuits for SIMPLIS compatibility.

The parameterization system includes conditional and looping features using the controls .IF and .WHILE.

Launching Preprocessor

The preprocessor is launched using the script command PreProcessNetlist. The syntax is:

PreProcessNetlist [/inAppend *extraInputLines*] [/simulator SIMPLIS|SIMetrix] *inFile outFile*

where

inFile	Input file name to be processed
outFile	File to receive result
extraInputLines	Additional lines appended to input file. Each line separated by a semi-colon ';'.

The /simulator switch allows the specification of the simulator that the netlist is intended for and affects library searching and the effect of the .SIMULATOR control. The default value is "SIMPLIS".

The PreProcessNetlist command called automatically when a SIMPLIS simulation is initiated and the user does not usually need to be aware of this.

Library Search

The SIMetrix library will be searched for any device referenced in a subcircuit call that is not present in the input netlist. Only devices that are explicitly designated as SIMPLIS models will be recognized in this search. (Unless "/simulator SIMetrix" is specified at the PreProcessNetlist command line). SIMPLIS models are identified in the model files using the .SIMULATOR control. The syntax of this control is:

.SIMULATOR SIMPLIS | SIMetrix

The .SIMULATOR control applies to all models that appear after it until the next .SIMULATOR control.

If the device is found in the library, its text will be entered as if it had appeared in the input netlist. Currently only subcircuit devices are resolved in this library search. All primitive devices defined using .MODEL must be defined in the input netlist.

Note also that currently, only the SIMetrix global library will be searched. The .LIB control is not supported.

Parameters

Parameters may be defined using the .VAR control which has the following syntax:

.VAR *parameterName*={ *parameterExpression* }

parameterName may be any alphanumeric sequence and must start with a letter or underscore. *paramterExpression* may be any valid SIMetrix expression as detailed in the script reference manual. String expressions are acceptable. parameterExpression may reference parameters defined in earlier .VAR controls.

Any part of the netlist following a .VAR control may contain parameter expression enclosed with curly braces. E.g.

```
.VAR resval = { 1K }

R1 1 2 {resval * 2}
```

.VAR may be used inside subcircuit definitions in which case the have local scope. This means that the parameter definition is only valid within that subcircuit definition. Note that this scope is not inherited by nested subcircuit definitions. However, parameters defined at the top level have global scope.

Passing Parameters to Subcircuits

Parameters may be passed to subcircuits via the subcircuit call. The syntax is:

X*xxx nodes subname* vars: *param1=value1 param2=value2 ...*

value1, 2 may be constants or an expression enclosed in curly braces. The expression may use values defined in previous .VAR controls and, if the X line is itself inside a subcircuit definition, values passed to that definition. The expression may not however, reference other passed parameters on the same X line. E.g. *value2* may not reference *param1*.

Conditional Lines

The .IF control may be used to define lines that are passed to the output only if specified conditions are met. The syntax is as follows:

.IF {*expression*}

netlist lines

...

[.ELSE

netlist lines

...]

.ENDIF

If *expression* resolves to a non-zero value then the lines up to .ELSE will be output otherwise the lines between .ELSE and .ENDIF will be output. The .ELSE is optional. .IF/.ELSE/.ENDIF may be nested to any level.

expression may be any valid arithmetic expression and may refer to previously defined parameters including those passed through a subcircuit call.

Looping

The preprocessor may be instructed to output a repeated sequence of lines using the .WHILE control. The syntax is:

.WHILE {*expression*} [*max_loop_count*]

netlist lines

...

.ENDWHILE

The block between .WHILE and .ENDWHILE will be repeated as long as *expression* is non-zero up to a maximum of *max_loop_count* times. If *max_loop_count* is omitted it takes a default value of 100. *max_loop_count* is intended as a safety measure to prevent an endless loop from filling the user's fixed disk to its capacity.

Running Monte Carlo and Multi-step Analyses

The SIMetrix environment has facilities to run multiple SIMPLIS analyses. Facilities to sweep parameter values and to randomly assign parameters for Monte Carlo are provided.

For more information, please refer to the SIMetrix *User's Manual*.

Chapter 8 Simplis Data Files

Overview

As SIMPLIS carries out an analysis, it creates additional data files in the same directory as the input file. These files are created by SIMPLIS primarily for its own use. It is recommended that everyone browse through this chapter to get an idea what these data files accomplish and as a result, be able to take advantage of these data files during various analyses. The section “Taking Advantage of Existing Files” on page 115 is particularly helpful for running time-domain simulations.

In the illustration that follows, the name of the input file is assumed to be "XXXX". Each data file generated by SIMPLIS is named "XXXX.extension" where "extension" is a string of characters particular to the data file.

The version of SIMPLIS supplied with SIMetrix sends its simulation data to SIMetrix which is then responsible for saving it. This data is usually stored in .sxdat files located in the TEMPDATA directory.

The Listing Data File

The listing file is automatically generated by SIMPLIS. It is named

XXXX.lst

where "XXXX" is the name of the input file to SIMPLIS. Ordinarily, this listing file is a rehash of the input file in a more organized manner. If you have elected to specify the EXPAND option in the input file, additional information illustrating how the subcircuit calls are being expanded will be written to this listing file.

Error Message Data File

The error message data file is named

XXXX.err

where "XXXX" is the name of the input file to SIMPLIS. If there is a syntax error detected in the input file or if a simulation error is encountered, error messages will be recorded in this error message file. You can examine this error message file and other data files generated by SIMPLIS to determine the cause of the problem.

SIMetrix automatically displays in the command shell the contents of this file if it exists.

The "State of Exit" Data File

At the end of a time-domain simulation, SIMPLIS always writes out the state of the system under study to a "State of Exit" file. This data file contains
the voltage across each linear capacitor or piecewise-linear capacitor,

1. The current through each linear inductor or piecewise-linear inductor,
2. The state of each simple switch or simple transistor switch,
3. The segment of operation of each piecewise-linear resistor, and
4. The output logic state of each simple logic gate

The State of Exit data file is written out only if a time-domain analysis has been specified. Traditionally, this file is named

```
XXXX.init
```

where "XXXX" is the name of the input file to SIMPLIS. The "State of Exit" file has been given the file extension ".init" because it is written in a format such that it can be easily merged into the input file to provide initial condition(s) for a continued simulation.

Suppose a simulation has been carried out for 200 microseconds and after examining the data you have decided to run for another 200 microseconds. One approach is to repeat the first simulation with the original initial conditions by changing the run time to 400 microseconds. This approach has the disadvantage of repeating the simulation of the first 200 microseconds. On the other hand, if you are not particularly interested in the waveforms of the first 200 microseconds, the ".init" file can be used to override the initialization provided in the input file by including the contents of this data file in the input file. In such a case, the run time of the second simulation only has to be carried out for 200 microseconds as the ".init" file already contains the state of the system at the end of the first simulation, which is at t = 200 microseconds.

Switching Instance Data File

When a transient analysis is run, the control statement

```
.TRAN tstop tsave
```

instructs SIMPLIS to carry out a time-domain simulation from t=0 to t=tstop.
No data will be generated or saved until the simulation reaches t=tsave. Starting at t = tsave, every switching instant is recorded in a data file called the "Switching Instance" data file. This data file contains a description of the system under study and a snap shot of the system at every switching instance from tsave to tstop, inclusively. This data file is named

XXXX.t1

where "XXXX" is the name of the input file to SIMPLIS. The Switching Instance data file is only produced for a time-domain analysis

Time-domain Data Output

During a time-domain simulation, SIMPLIS reads the Switching Instance data file and reconstructs detailed waveforms from the data stored in the Switching Instance data file. Detailed waveforms are reconstructed for various print variables for the time interval from the value specified for PSP_START to the value specified for PSP_END. You are referred to “Option Statements” on page 95 on the usage of option statements

to specify values for these two parameters. The waveform data are sent to the SIMetrix front end which is responsible for storing and if required plotting the data. This Time-Domain data is only produced for a time-domain analysis and only if PSP_NPT is specified as an option in the input file.

SIMPLIS also generates a file with the extension .T2 which contains details of the data stored but not the actual data itself. See example below.

```
$$$ 1 1006 6simplis DATA FOR PLOTTING, Mon Nov 25 14:00:17
INPUT FILE: demo01
Transient Analysis
7 DATA PTS
3 VARIABLES
0 TIME
1 V(4,0)
2 I(LL)
3 V(CC)
```

The Topology Information File

During a time-domain simulation, if the system under study spends a non zero amount of time in a certain circuit topology, SIMPLIS will save the important information associated with this particular topology in a file called the Topology Information file. In addition, this file also contains a description of the system under study and it is named

XXXX.tc

where "XXXX" is the name of the input file to SIMPLIS.

Taking Advantage of Existing Files

Often, you will make multiple time-domain simulation runs on the same system. After the first simulation run is finished, the Switching Instance Data file mentioned in “Switching Instance Data File” on page 114 and the Topology Information file outlined above are created. After examining the waveforms from this simulation run, you may decide to either repeat the simulation with a longer run time or repeat the same simulation run with a different set of print variables. In either case, there is no change in the system under study and the Switching Instance Data file and the Topology Information file contain valuable information associated with the time-domain simulation. In the next simulation run, SIMPLIS can take advantage of the data stored in these two files and reduce the computation time in the simulation.

Every time SIMPLIS is invoked, it looks for the presence of the Topology Information file, the "XXXX.tc" file. If this file is present and the circuit it describes matches that of the input file, SIMPLIS can take advantage of the existing knowledge on the different circuit topologies and the overhead spent in analyzing each known circuit topology is reduced.

As outlined in “Overview” on page 8, SIMPLIS is a two-pass simulator when it comes to time-domain simulation. The first pass of the simulation is the regular simulation and it computes the state of the simulation at each switching instance and saves the data in the Switching Instance Data file -- the "XXXX.t1" file. In the second pass,

called the Post-Simulation Processing, or "PSP" for short, detailed waveform data are reconstructed from the data saved in the Switching Instance Data file. Every time SIMPLIS starts a time-domain analysis, it looks for the presence of the Switching Instance Data file. If this file is present, SIMPLIS checks to see if it is usable. This file is considered usable if it describes the same circuit and the same analyses as defined in the input file. If the Switching Instance Data file is considered usable, SIMPLIS can skip the first pass of the simulation and directly go to the Post-Simulation Processing phase, saving a substantial amount of simulation time. Changing the PSP_START, PSP_END, and PSP_NPT parameters through the option statements is not considered to be a change in the analyses as these parameters only affect the PSP-phase of the simulation.

Since the Topology Information file and the Switching Instance Data file contain critical data of the simulation, they have been created with a read-only protection mode to prevent accidental changes being made to them.

Switching Instance Data File for the POP Analysis

When the periodic operating point analysis is applied to a system, a data file is generated, named

XXXX.t3

where "XXXX" is the name of the SIMPLIS input file. The periodic operating point analysis is discussed in detail in Chapter 10. This data file contains a description of the system under study and a snap shot of the system at every switching instance during a periodic operating point analysis. While this file will not be directly used by you, it is used by SIMPLIS to generate print/plot file for the Periodic Operating Point Analysis described in the next section. As a result, this data file is created with a read-only protection mode.

Data for the Periodic Operating Point Analysis

POP analysis creates data in exactly the same manner as for time-domain transient analysis. See “Time-domain Data Output” on page 114.

Print/Plot File for Frequency-Domain Analysis

When the frequency-domain small-signal analysis is carried out, the resulting data are sent to SIMetrix in the same way as for transient analysis. The only difference is that small-signal analysis data is complex.

SIMPLIS also creates a .F2 file for small-signal analysis with a similar format to the .T2 file. See “Time-domain Data Output” on page 114.

Chapter 9 Simplis-TX Examples

Overview

Examples are provided in this chapter to help the user get familiar with the syntax of the input file of SIMPLIS and understand various features of SIMPLIS-TX. Examples from using SIMPLIS-POP and SIMPLIS-FX are given in Chapter 10 and Chapter 11 respectively. The system studied in these examples represent a variety of systems encountered in power electronics. Each example is intentionally restricted to be small to make the illustration concise. For the same reason, the device models in this example have been kept simple but functionally adequate. These simple device models serve as the basic building blocks from which more complex models can be formulated.

All the examples in this chapter are installed as "ready to run" SIMetrix schematics under the directory *root*\Work\Examples\SIMPLIS\Manual_Examples. The input files shown in this chapter are in fact the simulation decks as created by SIMetrix but with plotting statements removed.

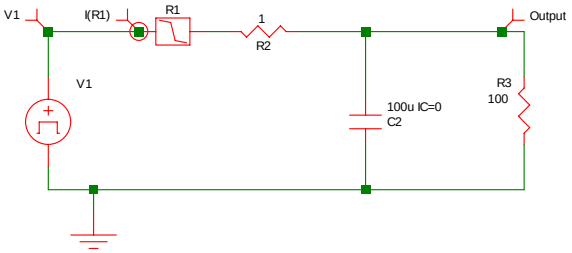
Example 1 -- Rectifier with RC load

The system under study in this example is the simple rectifier circuit shown in Figure 9.1. The ac voltage source V1 is a 154-V peak-to-peak 60 Hz sinusoidal voltage source. The variables of interest are the voltage across the ac voltage source, the voltage across the filter capacitor, and the current through the diode, which is modeled by the piecewise-linear resistor R1 (!R$R1 in the input file). The input file for the circuit of Example 1 is shown in Figure 9.2.

The time-domain simulation is carried out for 84 milliseconds, which is slightly longer than five complete cycles of the 60 Hz ac source. The parameter PSP_NPT is set to 211, instructing SIMPLIS to create the Time-Domain Print/Plot File with the print variables V(V1), V(C2), and I(R1) and a time resolution of
[(84) / (211 _ 1)] = 0.4 msec.

Waveforms obtained from this simulation are shown in Figure 9.3.

R1 is a PWL resistor designed to
have the characteristics of a diode



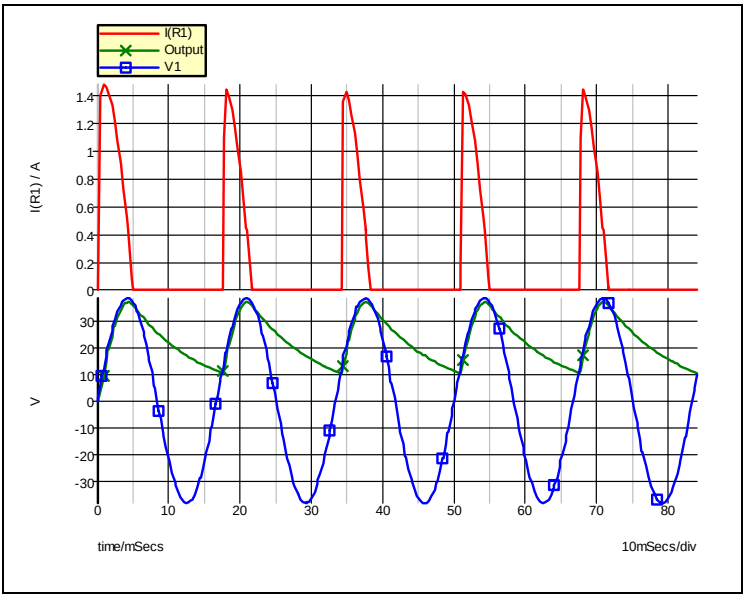
9.1 Example 1: Rectifier with RC Load

```
* Single RC Source Rectified into a RC Low-Pass Filter
.PRINT  ALL
.OPTIONS  PSP_NPT=211
.TRAN 84m 0

V1 1 0 SIN VOFFSET=0 APEAK=38.5 FREQ=60 TDELAY=0
+ OFF_UNTIL_DELAY=NO DAMP_COEF=0
C2 3 0 100u IC=0
!R$R1 1 2 R1$TP_SSPWLR IC=1
.MODEL R1$TP_SSPWLR VPWLR NSEG=2 X0=0 Y0=0 X1=1 Y1=0.5U
+ X2=1.1 Y2=1
R2 3 2 1
R3 0 3 100

.END
```

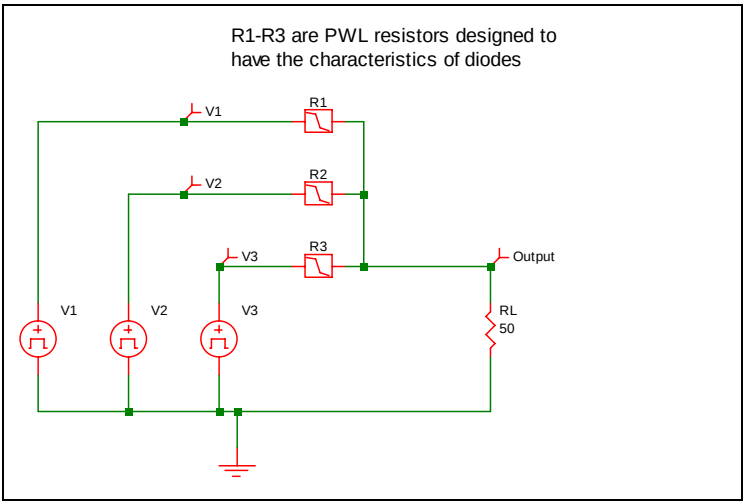
9.2 SIMPLIS Input File for Example 1 (as generated by SIMetrix)



9.3 Waveforms for Example 1

Example 2 -- 3-Phase Rectifier with Resistive Load

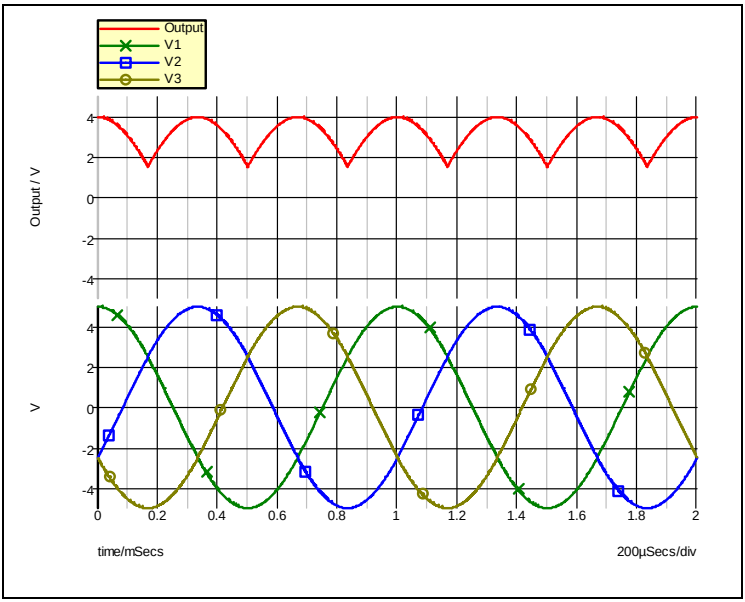
The circuit for Example 2 is a three-phase rectifier with resistive load, as shown in 9.4. It is made up of a three-phase ac voltage source in a Y-configuration which is rectified into a single load connected to the neutral line. The three voltage sources are modeled as cosinusoidal voltage sources of the same frequency with delays equal to 0, 1/3, and 2/3 of the period. The variables of interest are the three line-to-neutral voltages V(1), V(2), and V(3), and the load voltage V(4). The SIMPLIS input file for the circuit of Example 2 is shown in 9.5 . Waveforms for these four variables are plotted in 9.6.



9.4 Three-phase Rectifier with Load

```
* 3-Phase AC Voltage Source In A Y-Configuration
.PRINT  ALL
.OPTIONS  PSP_NPT=1001
.TRAN 2m 0
V1 1 0 COS VOFFSET=0 APEAK=5 FREQ=1k PDELAY=0
+ OFF_UNTIL_DELAY=NO DAMP_COEF=0
V2 3 0 COS VOFFSET=0 APEAK=5 FREQ=1k PDELAY=120
+ OFF_UNTIL_DELAY=NO DAMP_COEF=0
V3 4 0 COS VOFFSET=0 APEAK=5 FREQ=1k PDELAY=240
+ OFF_UNTIL_DELAY=NO DAMP_COEF=0
RL 2 0 50
!R$R1 1 2 R1$TP_SSPWLR IC=1
.MODEL R1$TP_SSPWLR VPWLR NSEG=2 X0=0 Y0=0 X1=1 Y1=5u
+ X2=1.1 Y2=1
!R$R2 3 2 R2$TP_SSPWLR IC=1
.MODEL R2$TP_SSPWLR VPWLR NSEG=2 X0=0 Y0=0 X1=1 Y1=5u
+ X2=1.1 Y2=1
!R$R3 4 2 R3$TP_SSPWLR IC=1
.MODEL R3$TP_SSPWLR VPWLR NSEG=2 X0=0 Y0=0 X1=1 Y1=5u
+ X2=1.1 Y2=1
.END
```

9.5 Input File for Example 2 (as generated by SIMetrix)

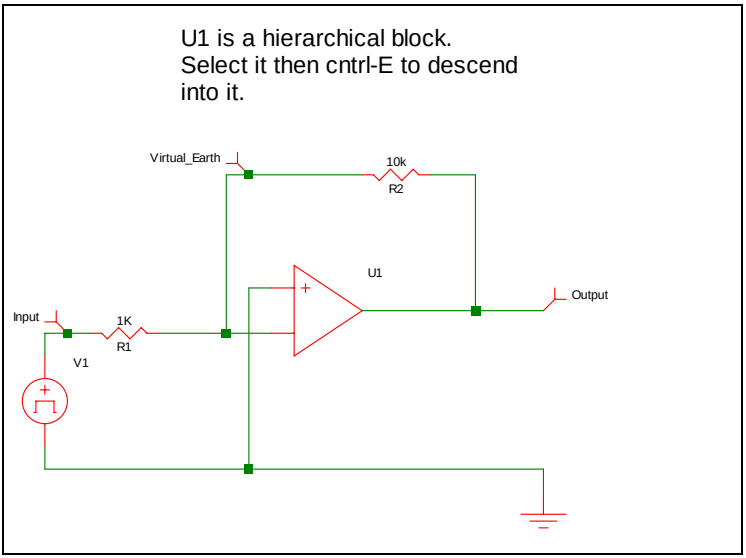


9.6 Waveforms for Example 2

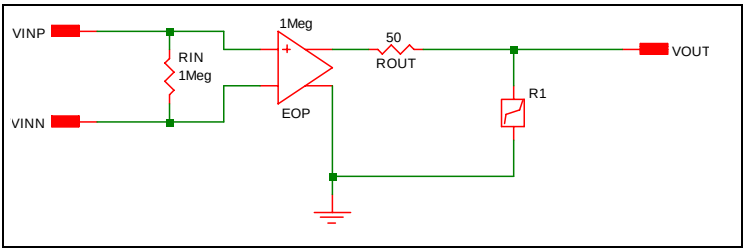
Example 3 -- Operational Amplifier with Saturation

The system under study in this example is a simple operational amplifier circuit driven to saturation by a sinusoidal input voltage. 9.7 (a) represents the circuit and 9.7 (b) is the piecewise-linear model of the system.

The operational amplifier is modeled by the input resistance RIN between its differential inputs, the voltage-controlled voltage source EOP, the output resistance ROUT, and the piecewise-linear resistor !RSAT. The purpose of the PWL resistor R1 (!R$R1 in the input file) is to model the saturation of the operational amplifier whenever the output voltage rises above 5V or drops below -5V. Placing the PWL resistor R1 across the output of the opamp is one of many possible ways to model the saturation of an operational amplifier. The variables of interest are the input sinusoidal voltage, the voltage across the differential inputs of the opamp, and the output of the opamp. The SIMPLIS input file defining the piecewise-linear model for the circuit of Example 3 is given in 9.8 and waveforms obtained from this simulation are shown in 9.9 .



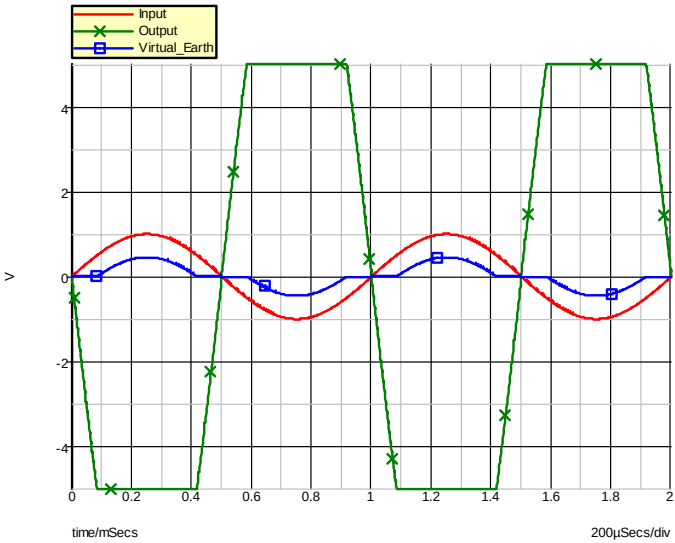
9.7a Example 3 Operational Amplifier with Saturation Circuit Diagram



9.7b Example 3 Operational Amplifier with Saturation - Piecewise-linear Equivalent Circuit

```
* Saturation of an Operational Amplifier
.PRINT  ALL
.OPTIONS  PSP_NPT=1001
.TRAN 2m 0
X$U1 2 0 1 opamp
V1 3 0 SIN VOFFSET=0 APEAK=1 FREQ=1k TDELAY=0
+ OFF_UNTIL_DELAY=NO DAMP_COEF=0
R1 1 3 1K
R2 2 1 10k
.SUBCKT opamp  3 4 2
.NODE_MAP VINN 2
.NODE_MAP VINP 4
.NODE_MAP VOUT 3
RIN 4 2 1Meg
EOP 1 0 4 2 1Meg
ROUT 3 1 50
!R$R1 3 0 R1$TP_SSPWLR IC=1
.MODEL R1$TP_SSPWLR VPWLR NSEG=3 X0=-5.1 Y0=-1MEG
+ X1=-5.0 Y1=-1U X2=5.0 Y2=1U X3=5.1 Y3=1MEG
.ENDS opamp
.END
```

9.8 Input File for Example 3



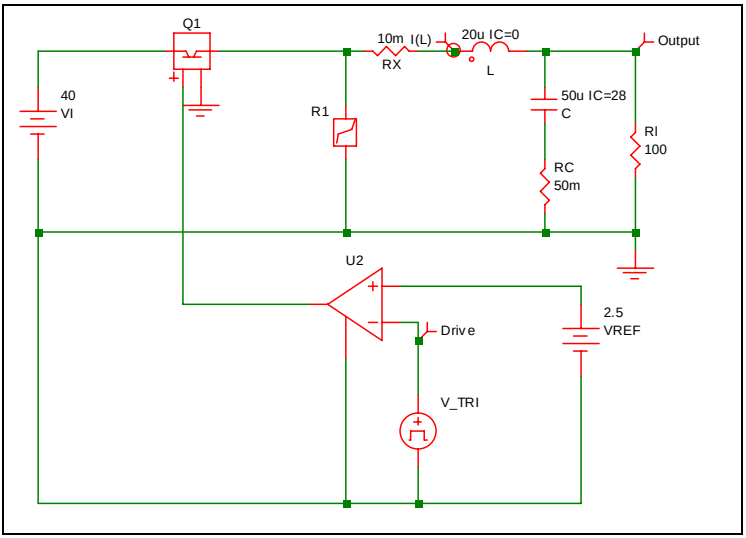
9.9 Waveforms for Example 3

Example 4 -- Unregulated Converter

Shown in 9.10 is the schematic of a simple current step-up (buck) converter operating under a fixed-frequency control law. This converter is not regulated by closed-loop control and the duty ratio of the transistor Q1 is determined by comparing the fixed reference voltage of 2.5 V to the output of the triangular function generator. The SIMPLIS input file describing this converter is shown in 9.11

Since the transistor Q1 is driven to behave like a controlled switch, it is modeled by a simple transistor switch with the LEVEL parameter set to 1. For the rest of the power stage of this converter, the diode is modeled by the piecewise-linear resistor R1 (!R$R1 in the input file), the energy -storage inductor by ideal inductance L and winding resistance RX, the output capacitor by ideal capacitor C and equivalent series resistance RC, and the load by the resistance RL.

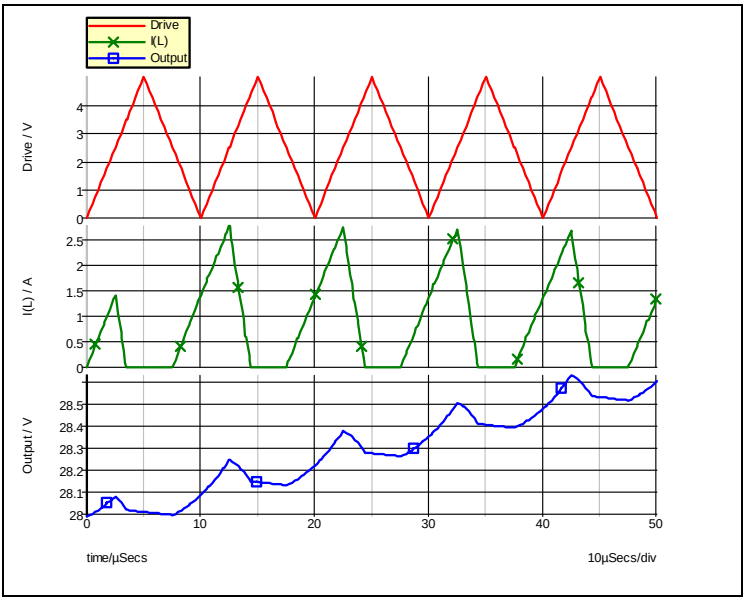
The current through the inductor L is initialized to a value of zero at the start of the simulation to represent operation of the converter in the discontinuous-mmf mode. Waveforms obtained from this simulation are displayed in 9.12 .



9.10 Unregulated Converter

```
* Fixed-Frequency Unregulated Current Step-Up Converter
.PRINT  ALL
.OPTIONS  PSP_NPT=201
.TRAN 50u 0
X$U2 6 0 8 9 SIMPLIS_COMP$1
V_TRI 9 0 TRI V1=0 V2=5 FREQ=100k DRATIO=500m DELAY=0
+ OFF_UNTIL_DELAY=NO
VREF 8 0 2.5
C 5 7 50u IC=28
L 4 5 20u IC=0
VI 2 0 40
RC 7 0 50m
Rl 5 0 100
Q1 2 3 6 0 Q1$TP_VCQ IC=CLOSE
.MODEL Q1$TP_VCQ VCQPOS VSAT=700m RSAT=100m ROFF=10Meg GAIN=10
+ TH=2.5 HYSTWD=100u LOGIC=POS LEVEL=1
!R$R1 0 3 R1$TP_SSPWLR IC=1
.MODEL R1$TP_SSPWLR VPWLR NSEG=2 X0=0 Y0=0 X1=0.7 Y1=10U
+ X2=0.8 Y2=1.00001
RX 4 3 10m
.SUBCKT SIMPLIS_COMP$1  201 100 101 102
!DCOMP 201 100 101 102 MCOMP IC=1
.MODEL MCOMP COMP RIN=1e+007 ROUT=50 VOL=0 VOH=5
+ HYSTWD=1e-006 DELAY=0
.ENDS SIMPLIS_COMP$1
.END
```

9.11 Input file for Example 4



9.12 Waveforms for Example 4

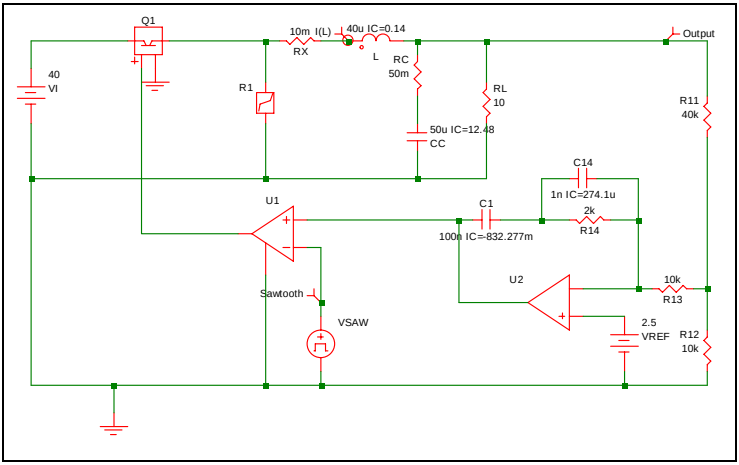
Example 5 -- Regulated Converter

The diagram shown in fig. 9.13 is the schematic of a regulated current step-up (buck) converter operating under a fixed-frequency control law.

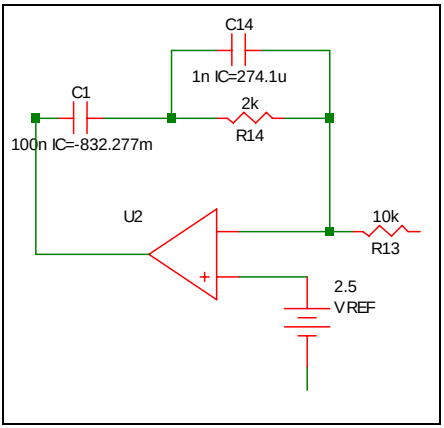
The modeling of the switching transistor, the diode, the energy-storage inductor, the output-filter capacitor, and the load of the power stage is similar to the modeling of the corresponding components in Example 4 and is not elaborated on here. Again, with the transistor modeled as a simple controlled switch, the base-drive shown in 9.13 does not need to be modeled in the input file as the output of the comparator can directly control the simple transistor switch.

The error amplifier in the controller is shown separately in 9.14 (a) and its piecewise-linear equivalent is shown in 9.14 (b). The operational amplifier is modeled with an input resistance RIN between its differential inputs and the voltage- controlled voltage source EOP at its output. 9.14 illustrates how an opamp circuit can be modeled by a simple network if the opamp is not driven into saturation. More sophisticated models for the operational amplifier can be built upon this basic model.

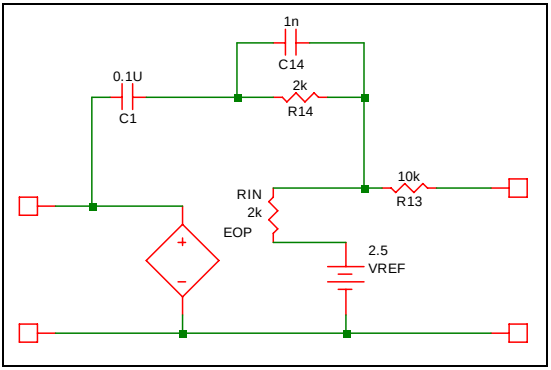
The input file describing this converter is shown in 9.15. In this example, we are interested in the steady-state waveforms of the sawtooth voltage V(11), the voltage V(RL) across the load and the current I(L) through the inductor.



9.13 Example 5 Regulated Converter



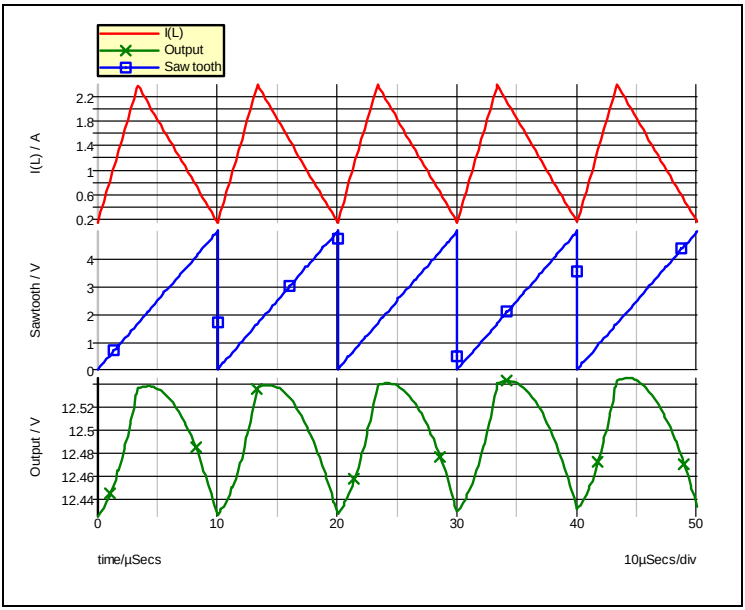
9.14a Error Amplifier Circuit



9.14b Piecewise-linear Equivalent

```
* Regulated Converter
.PRINT  ALL
.OPTIONS  PSP_NPT=201
.TRAN 50u 0
X$U2 11 13 10 opamp
VSAW 12 0 SAW V1=0 V2=5 FREQ=100k DELAY=0 OFF_UNTIL_DELAY=NO
X$U1 6 0 11 12 SIMPLIS_COMP$1
VREF 13 0 2.5
L 4 5 40u IC=0.14
VI 2 0 40
R12 0 8 10k
R13 8 10 10k
RC 7 5 50m
RL 5 0 10
R11 8 5 40k
C1 11 9 100n IC=-832.277m
R14 10 9 2k
C14 9 10 1n IC=274.1u
CC 7 0 50u IC=12.48
Q1 2 3 6 0 Q1$TP_VCQ IC=OPEN
.MODEL Q1$TP_VCQ VCQPOS VSAT=700m RSAT=100m ROFF=10Meg
+ GAIN=10 TH=2.5 HYSTWD=100u LOGIC=POS LEVEL=1
!R$R1 0 3 R1$TP_SSPWLR IC=1
.MODEL R1$TP_SSPWLR VPWLR NSEG=2 X0=0 Y0=0 X1=0.7 Y1=10U
+ X2=0.8 Y2=1.00001
RX 4 3 10m
.SUBCKT SIMPLIS_COMP$1  201 100 101 102
!DCOMP 201 100 101 102 MCOMP IC=1
.MODEL MCOMP COMP RIN=1e+007 ROUT=50 VOL=0 VOH=5
+ HYSTWD=1e-006 DELAY=0
.ENDS SIMPLIS_COMP$1
.SUBCKT opamp  2 3 1
.NODE_MAP VINN 1
.NODE_MAP VINP 3
.NODE_MAP VOUT 2
RIN 3 1 5Meg
EOP 2 0 3 1 1Meg
.ENDS opamp
.END
```

9.15 Input File for Example 5 (generated by SIMetrix)

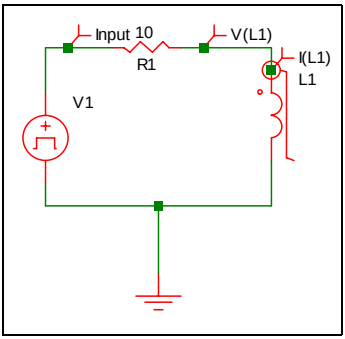


9.16 Waveforms for Example 5

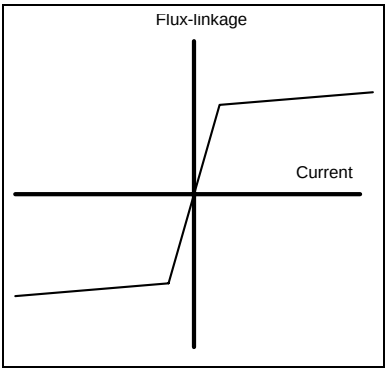
Example 6 -- Saturable Inductor

Depending on the system design, a magnetic element may be intentionally or unintentionally driven into saturation during transient or during steady-state operation. This example illustrates the usage of piecewise-linear (PWL) inductors in the input file. The system under study is shown in fig. 9.17 (a), which comprises of a sinusoidal voltage source driving a resistor in series with a saturable inductor. Fig 9.17 (b) shows the piecewise-linear flux linkage versus current characteristic of the saturable inductor. The input file for this circuit is shown in 9.18.

The saturable inductor is modeled by the piecewise-linear inductor L1 (!L$L1 in the input file), with the flux linkage entered in units of weber-turns. The variables of interest are the input voltage V(VI), plus the voltage V(L1) and the current I(L1) of the PWL inductor. The simulated waveforms for these variables are shown in fig 9.19.



9.17a The circuit diagram

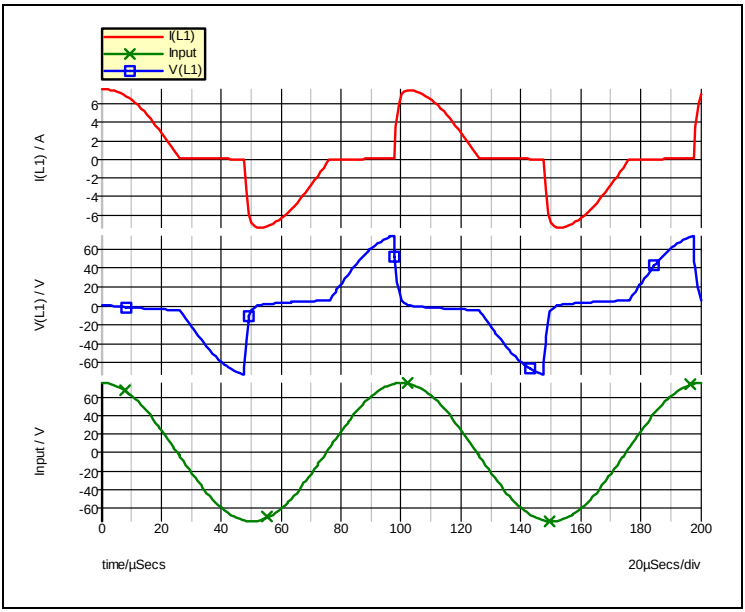


9.17b Flux-linkage vs. current characteristics of inductor

```
* Cosine Wave Driving A Resistor And A Saturable Inductor

VI 1 0 COS VOFFSET=0 APEAK=75 FREQ=10K
.PRINT  ALL
.OPTIONS  PSP_NPT=201
.TRAN 200u 0
V1 1 0 COS VOFFSET=0 APEAK=75 FREQ=10k TDELAY=0
+ OFF_UNTIL_DELAY=NO DAMP_COEF=0
!L$L1 2 0 L1$TP_SSPWLL
.MODEL L1$TP_SSPWLL PWLL NSEG=3 X0=-0.1 Y0=-500.5U
+ X1=-0.05 Y1=-500U X2=0.05 Y2=500U X3=0.1 Y3=500.5U
R1 2 1 10
.END
```

9.18 Input file for Example 6



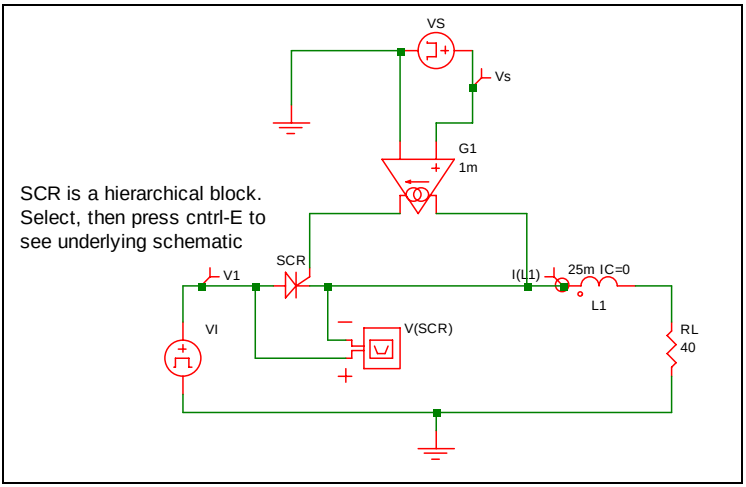
9.19 Waveforms for Example 6

Example 7 -- SCR with RL Load

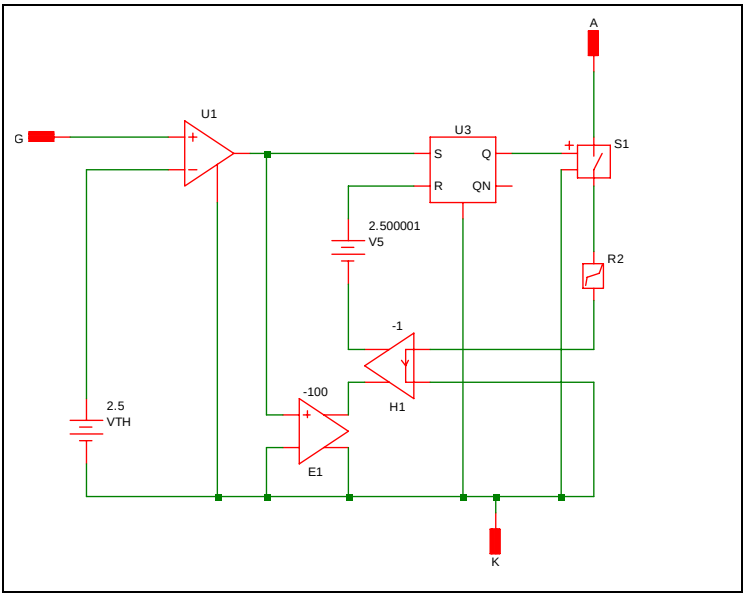
The system studied in this example, as shown in fig. 9.20, is a sinusoidal voltage source rectified into an R-L load through a silicon controlled rectifier.

In this example, the SCR is modeled by the series combination of the simple switch S1 and the piecewise-linear resistor !R2, as shown in fig 9.21. The input file for this circuit is shown in fig 9.22. The comparator U2 (!D$U2 in the input file), the SR Flip Flop U1 (!D$U1 in the input file), and the elements E1, H1, and V5 together form a network that models the switching of the SCR. When a sufficient gate current is applied between the gate and the cathode, causing the voltage across the non-inverting input of U2 to exceed 2.5 V, the output of U2 will rise to approximately 5 V, thus setting the SR Flip Flop, which in turn causes the switch S1 to be closed. When S1 is closed, the model characteristic from anode to cathode is the series combination of a small resistance of the switch and a diode. On the other hand, if the SR flip flop is reset, the switch S1 will be opened, and the model characteristic from anode to cathode looks like a large resistor in series with a diode. The flip flop is reset when the voltage across its reset input, which is equal to the sum of the voltages across E1, H1, and V5, exceeds 2.500001 V. This situation occurs when the gate signal is absent and the current through the SCR is negative. The presence of E1 makes sure that the reset input does not reach its threshold value of 2.500001 V whenever there is a gate signal present. E1 accomplishes this by having its output equal to 500 V whenever the output of U2 reaches 5 V.

The variables of interest are the input voltage V(VI), the signal voltage V(VS), the current I(L1) through the inductor, and the voltage V(SCR) across the SCR. A differential voltage probe has been added to plot the latter. The waveforms associated with these variables as obtained from the simulation are shown in fig. 9.23.



9.20 Example 7: A silicon-controlled rectifier with series R-L load

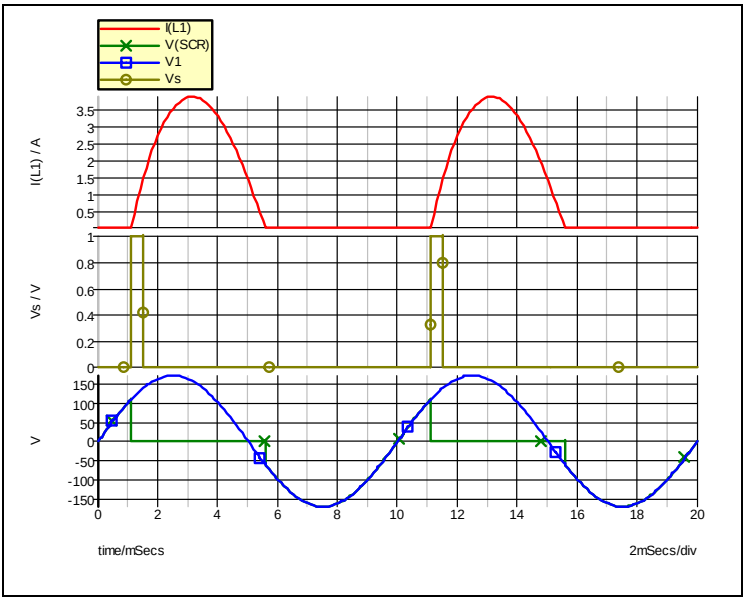


9.21 Piecewise-linear model of the SCR in terms of SIMPLIS elements

```
* Sine Wave Driving A R-L Load Through A SCR

.PRINT  ALL
.OPTIONS  PSP_NPT=201
.TRAN 20m 0
VS 1 0 PUL V1=0 V2=1 FREQ=100 T_RISE=0 T_FALL=0 PWIDTH=400u
+ DELAY=1.1m OFF_UNTIL_DELAY=NO
X$SCR 4 2 3 scr1
L1 3 5 25m IC=0
E$Probe2$TP_DIFFPRB 6 0 4 3 1
RL 5 0 40
G1 3 2 1 0 1m
.SUBCKT scr1  12 11 10
.NODE_MAP A 12
.NODE_MAP G 11
.NODE_MAP K 10
X$U3 2 5 10 1 4 SIMPLIS_SRFF$1
V5 4 7 2.500001
X$U1 1 10 11 3 SIMPLIS_COMP$2
H1 7 9 VH1$TP_CCVS -1
VH1$TP_CCVS 8 10 0
E1 9 10 1 10 -100
VTH 3 10 2.5
!R$R2 6 8 R2$TP_SSPWLR IC=1
.MODEL R2$TP_SSPWLR VPWLR NSEG=2 X0=0 Y0=0 X1=1 Y1=1U
+ X2=1.1 Y2=2.000001
S1 12 6 2 10 S1$TP_SSVSW IC=Open
.MODEL S1$TP_SSVSW VCSW ROFF=10Meg RON=10u LOGIC=POS
+ TH=2.5 HYSTWD=20u
.SUBCKT SIMPLIS_COMP$2  201 100 101 102
!DCOMP 201 100 101 102 MCOMP IC=0
.MODEL MCOMP COMP RIN=5000 ROUT=50 VOL=0 VOH=5
+ HYSTWD=0.002 DELAY=0
.ENDS SIMPLIS_COMP$2
.SUBCKT SIMPLIS_SRFF$1  201 202 100 101 102
!D_SIMPLIS_SRFF 201 202 100 101 102 MSRFF IC=0
.MODEL MSRFF SRFF RIN=1e+007 ROUT=50 VOL=0 VOH=5
+ HYSTWD=2e-006 DELAY=0  TH=2.5 LOGIC=POS
.ENDS SIMPLIS_SRFF$1
.ENDS scr1
.END
```

9.22 Input File for Example 7 (generated by SIMetrix)



9.23 Waveforms for Example 7

Chapter 10 Simplis-POP

Overview

In the analysis of a switching piecewise-linear system, the steady-state solution is essential. For example, in the study of the line/load regulation of a regulated switching power system, the relevant information is the steady-state load voltage over a range of line/load conditions. Although carrying important information in its own right, the transient information on how the system settles to the new steady-state under the new line/load condition is not the focus of such a study. To carry out such a study, the load voltage is measured after the system has settled to new steady-state operations under new line/load conditions. Depending on the damping and the regulation circuitry of the system, it may take the system hundreds to thousands of switching cycles before settling to a new steady-state operation after each change in the line/load condition. While carrying out such a study with a brute-force simulation is possible, it can be time-consuming. Hence, there is a need for a special analysis tool that can "accelerate" the convergence of the system towards its steady-state operating condition without going through the actual transient.

Another example where the steady-state operation of a switched piecewise-linear system is essential is in the study of the load transient of a regulated switching power system. In such a study, the transient experienced by the system, in response to a load change, is monitored. When this study is carried out through simulation, one must initialize the circuit simulation to the initial steady-state solution of the system before the load change is initiated. Without using a steady-state solution to start the simulation, the transient obtained will be different from the transient measured in the laboratory.

A special algorithm for speeding up the computation of the steady-state solution of switched piecewise-linear systems have been incorporated into SIMPLIS as a special analysis tool. While it is mathematically more challenging and computationally more intensive, the computation of the steady-state solution of a switched system is conceptually quite similar to finding the DC operating point of a non- switching system under only DC excitations. In both cases, we are interested in finding the operation of the system in the absence of an external stimulus. Since the term "Operating Point Analysis" has been traditionally and widely used as the name for a DC operating-point analysis, the name "Periodic Operating Point Analysis," (POP) is given to this special algorithm for speeding up the computation of the steady-state solution of switched piecewise-linear systems that are periodically driven or self-oscillating.

The Periodic Operating Point Analysis tool in SIMPLIS is able to speed up the convergence to the steady-state solution of a switched piecewise-linear system that is either self-oscillating or driven by one or more periodic sources that are commensurate in their periods. To invoke such an analysis, the user only needs to add a few lines in the input file. “Statements Relating to POP Analysis” on page 136 explains in detail the format of the input statements related to the Periodic Operating Point analysis. “Synopsis of the Periodic Operating Point Analysis” on page 143 explains what happens during a Periodic Operating Point analysis. After reading this section, a user will understand the internal workings of this analysis tool and, as a result, will be able to use the analysis tool more productively. An example in “Example of Applying the

POP Analysis Tool" on page 151 illustrates the application of the POP analysis to a closed-loop regulated switching power system

Statements Relating to POP Analysis

The statements relating to the Periodic Operating Point analysis can be all classified as control statements of the type defined in Chapter 6. An analysis statement to invoke the POP analysis and three option statements are associated with the Periodic Operating Point analysis.

.POP Statement for POP Analysis

The .POP statement instructs SIMPLIS to perform a Periodic Operating Point analysis on the system under study. Just like any other analysis statement, the .POP statement can only appear within the scope of definition of the main circuit. In addition, there can be no more than one .POP statement in an input file. The format for the .POP statement is

.POP TRIG_GATE=*gate_name* TRIG_COND={0_TO_1|1_TO_0}
+ MAX_PERIOD=*max_period*
TD_RUN_AFTER_POP_FAILS=*tran_after_pop_fail*

where

.POP	is the four-character keyword ".POP".
TRIG_GATE=	is the ten-character keyword "TRIG_GATE=".
gate_name	is the device name of a defined logic gate. This device is considered the triggering gate of the POP analysis. Together with the parameter value of TRIG_COND, it determines the condition that constitutes the start of a new switching cycle in the POP analysis.
TRIG_COND=	is the ten-character keyword "TRIG_COND=".
0_TO_1	is the six-character keyword "0_TO_1".
1_TO_0	is the six-character keyword "1_TO_0".
MAX_PERIOD=	is the eleven-character keyword "MAX_PERIOD=".
max_period	is a positive floating-point number assigned to the parameter MAX_PERIOD.
tran_after_pop_fail	Controls behaviour if the POP analysis fails to converge. This has three modes of operation as follows: *tran_after_pop_fail* = 0: Display error message then abort *tran_after_pop_fail* = -1: Start a transient analysis with a run time equal to 100 x *max_period* (see above) *tran_after_pop_fail* = *t*, *t>0*: Start a transient analysis with a run time equal to *t*

See "Behaviour of POP Analysis after POP Convergence Failure" on page 141 for more details of this feature.

The .POP statement must be the first analysis statement among all types of analyses statements specified in an input file. Upon reading the .POP statement, SIMPLIS performs a Periodic Operating Point analysis of the system. If the Periodic Operating Point analysis is successful, SIMPLIS will perform the next analysis specified in the input file. If the Periodic Operating Point analysis is not successful, SIMPLIS will print out an error message and exit.

Definition of the Start of A Switching Cycle

During a Periodic Operating Point analysis, SIMPLIS performs a sequence of time-domain transient analyses on the system. These transient analyses are invisible to the user. Each time-domain transient analysis is stopped when the operation of the system reaches the start of a new switching cycle. The user can define the condition constituting the start of a switching cycle through the TRIG_GATE and TRIG_COND parameters in the .POP statement. Let us examine the two .POP statements below:

```
.POP TRIG_GATE=!D1 TRIG_COND=0_TO_1 MAX_PERIOD=100U

.POP TRIG_GATE=X1.!D1 TRIG_COND=1_TO_0
+ MAX_PERIOD=100U
```

The first .POP statement defines the start of a switching cycle as the time instant when the output of the logic gate named !D1 switches from logic 0 to logic 1. The second .POP statement defines the triggering gate as the logic gate named !D1 in the subcircuit referred to as X1 in the main circuit, and it defines the start of a switching cycle as the instant when the output of the triggering gate switches from logic 1 to logic 0.

The two .POP statements above are shown here for illustration purposes. Only one .POP statement can appear in a SIMPLIS input file. The second .POP statement also demonstrates that the triggering gate is not restricted to the logic gates defined in the main circuit. It can be any logic gate defined in the input file.

Definition of the Start of A Switching Cycle for a Driven System

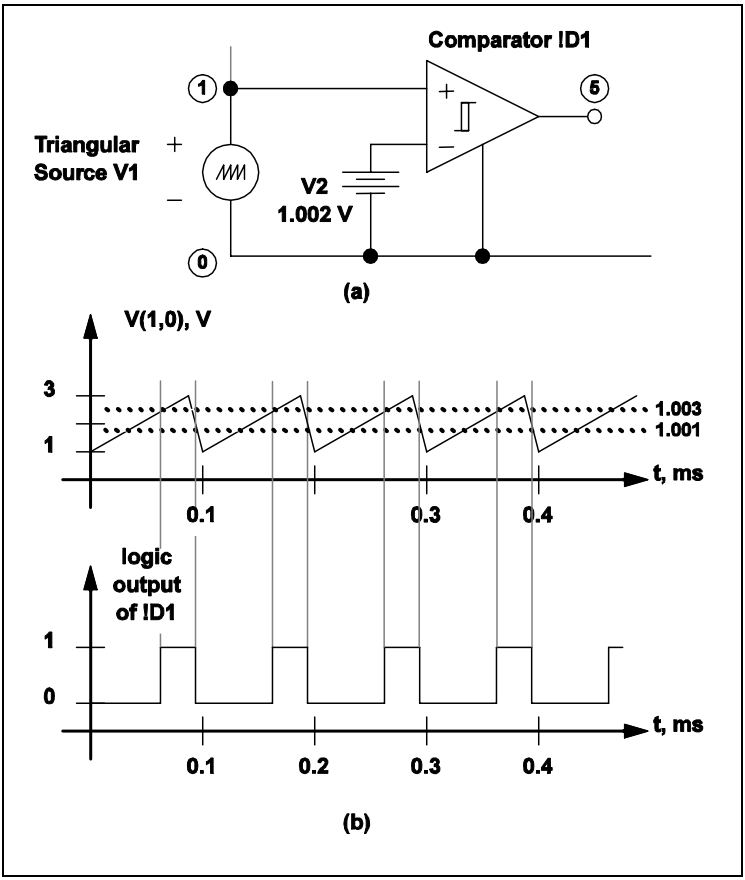
In a driven system, the driving periodic source is a good candidate to define the start of a switching cycle. Fig 10.1(a) shows a small section of a hypothetical circuit driven by a periodic triangular source V1 whose minimum and maximum voltages are, 1V and 3V, respectively. The triggering gate in this example is the comparator !D1 shown in the figure. Since V1 is periodic, the choice of the start of a cycle is arbitrary. For example, let us use the instant when the source value is decreasing and it is equal to 1.001V as the instant signifying the start of a switching cycle. The statements associated with the triangular source, the triggering gate, and the periodic operating point analysis for this example are:

```
V1 1 0 TRI V1=1 V2=3 FREQ=10K DRATIO=0.98
+ DELAY=0 OFF_UNTIL_DELAY=NO
!D1 5 0 1 2 M1 IC=0
.MODEL M1 COMP RIN=10MEG ROUT=8 HYSTWD= 2M
+ VOH=5 VOL=0
V2 2 0 DC 1.002
```

```
.POP TRIG_GATE=!D1 TRIG_COND=1_TO_0 MAX_PERIOD=300U
```

The model statement states that the output of the comparator is equal to logic 1 whenever the voltage at the non-inverting input is 1 mV above that of the inverting input, and that the output is equal to logic 0 whenever the voltage at the non-inverting input is 1 mv below that of the inverting input. Since the voltage at the inverting input of !D1 is a DC voltage source of 1.002 V, this implies that the output of !D1 is equal to logic 1 when the voltage of V1 exceeds 1.003 V, and it is equal to logic 0 when the voltage of V1 drops below 1.001 V. When the voltage of V1 is between 1.001 V and 1.003 V, the output state of !D1 remains unchanged. The diagram in fig 10.1(b) illustrates the relationship between the output state of !D1 and the voltage of source V1. The figure shows that the output of !D1 will have a logic 1 to logic 0 transition whenever the voltage of V1 is decreasing and equal to 1.001 V.

If there are multiple periodic sources driving the system, the periodic operating point analysis tool can be applied only if these sources are commensurate in their periods. In this case, the switching cycle should be defined with a period equal to the least common multiple of the periods of all periodic sources.



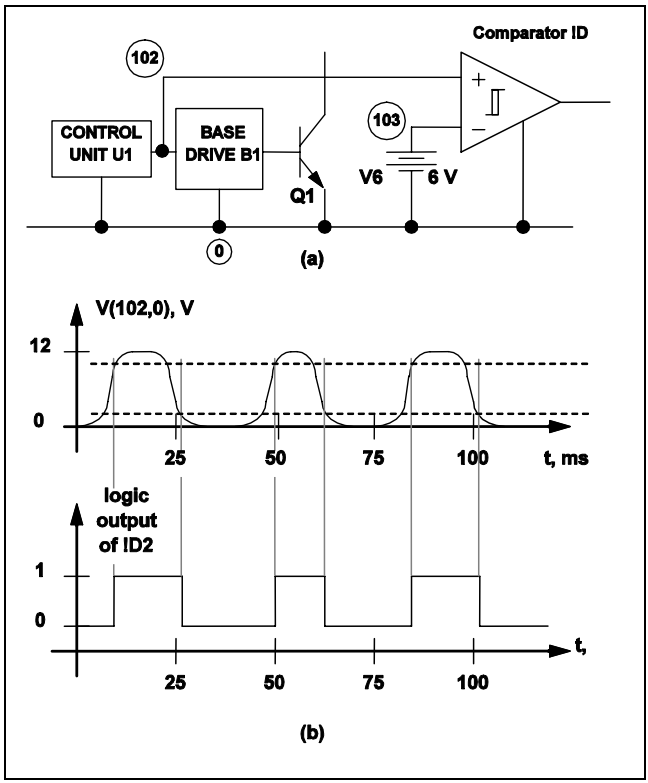
10.1 (a) A periodic triangular source, a battery, and a comparator used to define the start of a switching cycle, and (b) Output of the comparator in relationship to the differential voltage V(1,0).

Definition of the Start of A Switching Cycle for a Self-Oscillating System

In the case of a self-oscillating system, the start of a switching cycle should be defined to coincide with a major switching event such as the on/off switching of the main power transistors(s) of a self-oscillating converter. Fig 10.2 (a) shows a small section of a hypothetical self-oscillating switching system. The main switching transistor Q1 is controlled by the control unit U1 through the base drive circuit B1. The output level of U1 is equal to 12 V and 0 V, respectively, when it is trying to switch the transistor Q1 on and off. In this case, the start of a switching cycle is best defined in terms of the transition of the output of U1. The comparator !D2 in this figure serves this purpose and the input statements defining !D2 and the .POP analysis may look as follows:

```
!D2 101 0 102 103 M2 IC=0
.MODEL M2 COMP RIN=10MEG ROUT=8 HYSTWD=10
+ VOH=5 VOL=0
V6 103 0 DC 6
.POP TRIG_GATE=!D2 TRIG_COND=0_TO_1 MAX_PERIOD=200U
```

With these input statements, the output of the logic gate !D2 is equal to logic 1 and logic 0, respectively, when the output of U1 is above 11 V and below 1 V. When the output of U1 is between 1 V and 11 V, the output state of !D2 remains unchanged. Fig 10.2 (b) illustrates the relationship between the output state of U1 and the output state of !D2. From this figure, it can be deduced that the start of a switching cycle is now recognized to occur when the output of U1 is rising and equal to 11 V. The output of U1 as shown in Fig 10.2 (b) has finite-time transitions when it switches between 0 V and 12 V. If U1 is a SIMPLIS logic gate, we can use U1 instead of an additional logic gate as the triggering gate for the POP analysis.



10.2 (a) A sample circuit used to define the start of a switching cycle for a variable-frequency switching system, and (b) Output of the comparator !D2 in relationship to the waveform of the voltage V(102,0).

Definition of the Maximum Period

Depending on how the triggering gate and triggering condition are defined and depending on the capacitor voltages and inductor currents of the system, it is possible that the system either takes a very long time or is never able to reach a condition that triggers the start of a new switching cycle. Such a condition can occur at the start of the periodic operating point analysis or in the middle of a periodic operating point analysis. Since the triggering condition never occurs, one of the time-domain transient analyses mentioned in 10.2.1.1 may run forever. To avoid such a situation, the maximum simulation time for all POP time-domain transient analyses is set to the numerical value of the maximum period parameter. For example, a .POP statement such as

```
.POP TRIG_GATE=!D2 TRIG_COND=0_TO_1 MAX_PERIOD=500U
```

instructs SIMPLIS to carry out each of these transient analyses for a maximum duration of 500 microseconds. If the simulation time of a POP transient analysis reaches 500 microseconds without triggering the start of a new switching cycle, SIMPLIS will print out an error message and exit.

For a driven system, the maximum period should be set to at least three times the expected period. For a self-oscillating system, the maximum period should be set to about 10 times the longest period expected.

There are several reasons that SIMPLIS may fail to find a triggering condition for the system under study within the set maximum period. One is an error in the definition of the triggering condition. Given correctly defined triggering conditions, it is also possible that the transients resulting from the initial conditions specified for the system would prevent the system from reaching the triggering condition within the set maximum period. Usually, a regular time-domain transient analysis would reveal the reasons that the system failed to trigger the start of a new switching cycle.

Behaviour of POP Analysis after POP Convergence Failure

There are two choices of action that SIMPLIS will take when POP fails to converge. These are:

1. Abort the run and output an error message. The error is written to a file but will also be displayed in the command shell message box. This action is taken if the .POP parameter TD_RUN_AFTER_POP_FAILS is set to zero.
2. Start a time-domain (transient) analysis. The initial conditions for this time-domain analysis will be exactly the same as the initial conditions used for the POP run and will continue for a period controlled by the TD_RUN_AFTER_POP_FAILS parameter on the .POP line. See “.POP Statement for POP Analysis” on page 136 for details

   If a normal time-domain analysis had been specified to proceed after POP (for example, because a load transient study was being performed), then this time-domain analysis will proceed normally as if the POP analysis had converged successfully. This mode of operation makes it possible to study the load or line transient behaviour of systems for which POP convergence is difficult. Examples are systems running at very light load whereby they enter a pulse frequency mode of operation. Such systems do not have a well-defined steady state which makes POP convergence almost impossible. But by running a long enough time

domain analysis before applying load or line stimuli, a near-steady-state condition may be reached.

If no time-domain analysis had been specified after the POP analysis, the data generated during the additional time-domain analysis will be saved so that a diagnosis of the causes of POP convergence failure may be performed.

Options Associated with POP Analysis

There are five options associated with the periodic operating point analysis and they can be specified through the .OPTIONS statement explained in “Option Statements” on page 95. The options are specified in the form of

.OPTIONS opt1 opt2 ...

where opt1, opt2, and ..., are various options recognized by SIMPLIS. The five options associated with the periodic operating point analysis and their meanings are listed below.

POP_SHOWDATA — Shows the progress of the periodic operating point analysis. The default is not to show the progress. In general, this option is turned on as a debugging aid if a POP analysis fails.

POP_ITRMAX=*n* — Sets the maximum number of iterations for the periodic operating point analysis.

"POP_ITRMAX=" is the eleven-character keyword "POP_ITRMAX=".

n is a positive integer between 1 and 100. The default value for *n* is 20.

POP_CONVERGENCE=*value*

Sets the convergence criteria for the periodic operating point analysis. The convergence criteria is satisfied when the relative change in each state variable, between the start and end of a switching cycle, is less than this parameter.

"POP_CONVERGENCE=" is the eleven-character keyword "POP_CONVERGENCE=".

value is a positive floating point number between 1.0e-06 and 1.0e-14, inclusive. The default value for n is 1.0e-14.

Note: During the POP analysis, the maximum relative change in the state variables is reported in percentages, while the POP_CONVERGENCE parameter is entered in actual value.

POP_USE_TRAN_SNAPSHOT

This option instructs POP to take advantage of the last data point of a previous transient simulation, assuming the circuit and the initial conditions remained the same between the two simulation runs.

This option does not take on any value. It is either turned ON or turned OFF. If it is turned ON, the POP analysis will use the last data point of a previous transient simulation as the initial condition to start the POP analysis. Usually this leads to a faster POP analysis.

Example:

.OPTIONS POP_USE_TRAN_SNAPSHOT

POP_OUTPUT_CYCLES=n

Number of cycles of steady-state POP Data to show. After a successful POP analysis, SIMPLIS will generate the steady-state time-domain waveforms for an integral number switching cycles.

If set, the option value must be a positive integer between 1 and 16, inclusively.

If this option is not set, it defaults to 5.

Example:

.OPTIONS POP_OUTPUT_CYCLES=3

When the POP_SHOWDATA option is turned on, a print/plot file named "XXXX.t4", where "XXXX" is the name of the input file, is generated during the periodic operating point analysis. The format of this print/plot file is the same as that described for the "TIME-DOMAIN PRINT/PLOT FILE", or the "XXXX.t2" file, in “Time-domain Data Output” on page 114. It contains columns of all print variables versus the time variable. The user is reminded that the print variables are specified through the .PRINT statement as outlined in “Control Statements for Printing Variables” on page 101. The data in this data file can be plotted to reveal the progress of the POP analysis. More will be discussed on this data file in the next section.

The POP_ITRMAX option puts a limit on the number of iterations of the periodic operating point analysis that will be carried out by SIMPLIS. When the periodic operating point analysis fails to converge to a steady-state solution after this limit is reached, SIMPLIS prints out an error message and exits.

Synopsis of the Periodic Operating Point Analysis

As pointed out earlier, the periodic operating point analysis must be the first analysis specified in an input file. As a result, the initial capacitor voltages and initial inductor currents at the start of the periodic operating point analysis are the same as those specified in the device statements and any overriding initial conditions supplied in the .INIT statements. Then, the periodic operating point analysis tool carries out a sequence of time-domain transient simulation on the system. The actions of the periodic operating point analysis are summarized by the flow chart in Fig 10.3.

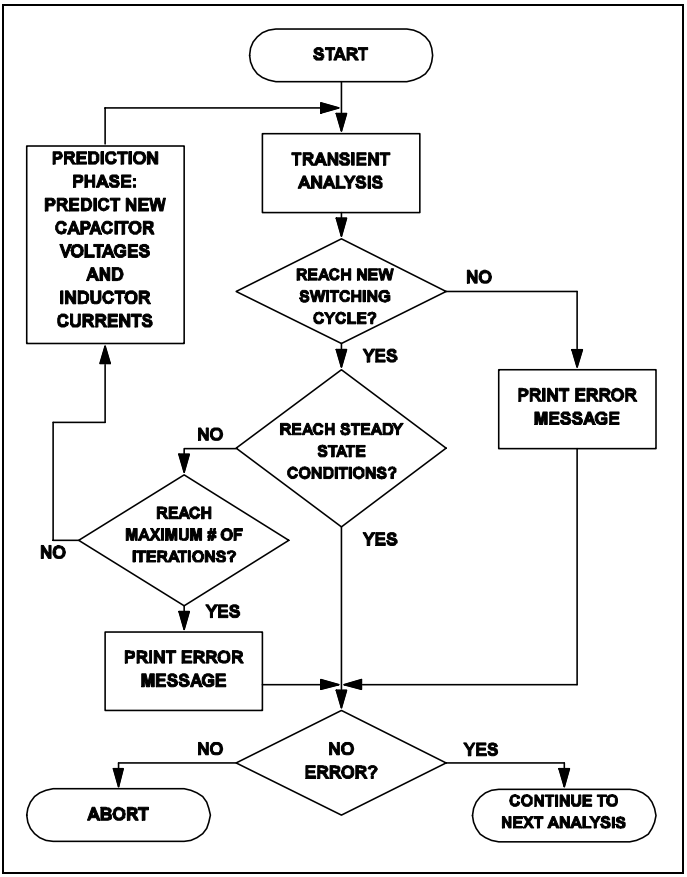
The transient analysis shown in Fig 10.3 is carried out indefinitely until the start of a new switching cycle is reached or until the value of the time variable has reached the parameter value of the maximum period set in the .POP statement. If the transient analysis fails to reach a condition to trigger the start of a new switching cycle before

the maximum period is reached, an error message similar to the one shown below is either printed on the screen or to an error file. After the error message has been printed, SIMPLIS aborts its execution.

```
Periodic Operating-Pt Analysis:
Reaching a time duration equal to
`3.00000e-04' without registering the
triggering condition that defines
the start of a period. Check your
circuit and/or initial conditions.
```

If the transient analysis is able to reach a condition triggering the start of a new switching cycle, the next task to be performed is to determine whether the system has reached steady-state operation. This is accomplished by finding the difference between the values of each capacitor voltage and inductor current at the start and end of the finished transient analysis. If the differences are small enough to be negligible for all capacitors and inductors, the system is considered to be in steady-state operation and the periodic operating point analysis is considered successful and it is stopped.

If the system is not considered to be in steady-state operation, the periodic operating point analysis tool applies a proprietary algorithm to predict what the values of the capacitor voltages and inductor currents at the start of a switching cycle ought to be had steady-state operation condition been reached. After this prediction phase, another transient analysis is initiated and the whole algorithm is repeated until either steady-state condition has been reached or the maximum iteration limit as set in the POP_ITRMAX option has been reached. If the maximum iteration limit is reached, an error message similar to the following is either printed on the screen or to an error file and execution of SIMPLIS is halted.



10.3 Actions carried out by the periodic operating point analysis tool

```
Periodic Operating-Pt Analysis:
Unable to find a periodic operating
point after 20 attempts. Check your
input file for errors. A change in
the initial condition may be necessary.
```

During each iteration, or pass, of the algorithm shown in Fig 10.3, the periodic operating point analysis tool provides a glimpse of the progress of the analysis by printing out short messages on the screen. The messages on the screen from a typical successful periodic operating point analysis will be similar to the following:

```
PERIODIC OPERATING-POINT ANALYSIS
PASS 1: 6.545582e+00
PASS 2: 2.848722e-01
PASS 3: 5.397469e-02
```

```
PASS 4: 5.006216e-03
PASS 5: 6.267872e-05
PASS 6: 1.004039e-08
PASS 7: 1.173411e-14
```

The number next to each pass index represents a measure of the maximum percentage difference between the values of each capacitor voltage and inductor current at the start and end of the corresponding transient analysis. Thus, at the end of the 4th transient analysis in this example, the error between the initial and final state vectors is less than 0.005%. The periodic operating point analysis tool considers steady-state operation has been reached when this percentage difference is smaller than the convergence parameter POP_CONVERGENCE, which has a default value of 1x10 -14 , or 1x10 -12 %.

The Time Variable During POP Analysis

In the periodic operating point analysis, our major goal is to find the steady-state operation of the system. How the system reaches the eventual steady-state operation from the original initial state is not as important. In addition, the action carried out by the prediction phase shown in fig 10.3 does not correspond to any real operation experienced by the system. Hence it is impossible to assign a true value to the time variable during a POP analysis. With no other better choices and without any loss of generality, we can reset the time variable at the exit of a POP analysis to its original value at the entry of the POP analysis. Since the POP analysis is the first analysis allowed, this means that the time variable is artificially reset to 0.0 at the end of a POP analysis. Consequently, if there is a time-domain transient analysis immediately following a POP analysis, the time variable for that time-domain transient simulation will start at 0.0 and the time variable will advance forward as usual in the regular time-domain transient analysis.

How POP Deals with Time Varying Sources

At the entry of a periodic operating point analysis, all time varying sources are classified as either periodic or aperiodic. Sawtooth, triangular, square, and pulse sources are always considered to be periodic. A sinusoidal or cosinusoidal source is considered to be periodic if its damping coefficient is equal to 0.0, otherwise it is considered as aperiodic. Exponential pulse sources and piecewise-linear sources are always considered to be aperiodic. Periodic sources are left unchanged during a POP analysis, i.e. they remain as sources with time-varying source values. The treatment of the aperiodic sources during a POP analysis, however, are quite different. They are turned into DC sources during a POP analysis, with each source held constant at its value just before the entry of the POP analysis.

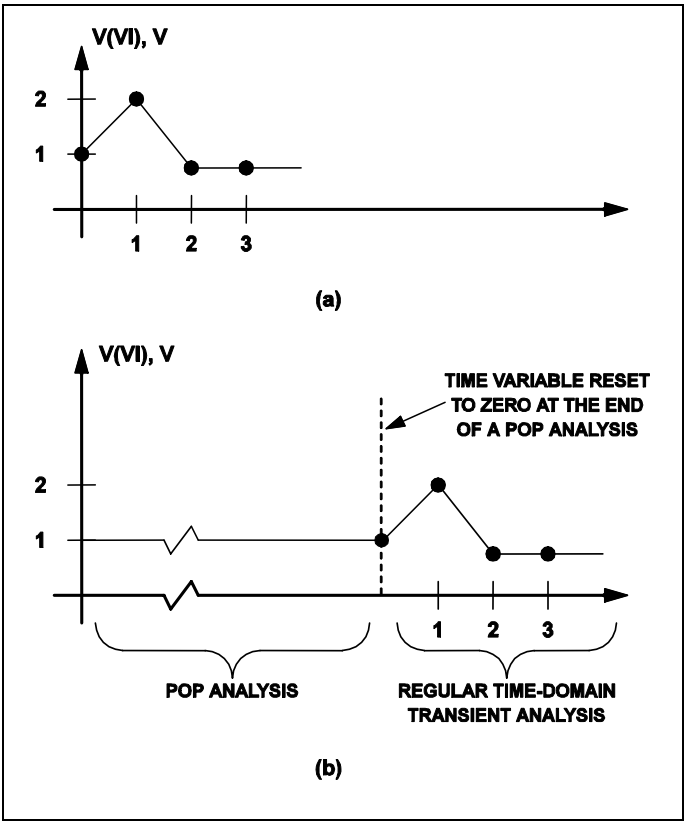
Aperiodic Sources

Aperiodic sources are held at a constant value during the POP analysis. This is necessary in order to find the periodic operating point. After the POP analysis has finished, the source values of the aperiodic sources are allowed to be time varying according to their definition in the input file. Take the piecewise-linear source defined in the following lines of statements as an example.

```
V1 101 0 PWL NSEG=3
+ X0=0.0 Y0=1.0
+ X1=1.0 Y1=2.0
```

```
+ X2=2.0 Y2=0.5
+ X3=3.0 Y3=0.5
```

Normally, the source value V(V1) would be as shown in fig 10.4 (a). When a POP analysis is carried out, the source value V(V1), during the entire POP analysis, is clamped by SIMPLIS at 1V, the value of the source at the beginning of the POP analysis. If there is a time-domain transient analysis immediately following the POP analysis, the waveform V(V1), during this time-domain transient analysis, will be as indicated in fig 10.4 (b): linearly rising from 1V to 2V during the first second, linearly decreasing from 2V to 0.5V during the next second, and remaining at 0.5V for the rest of the simulation.



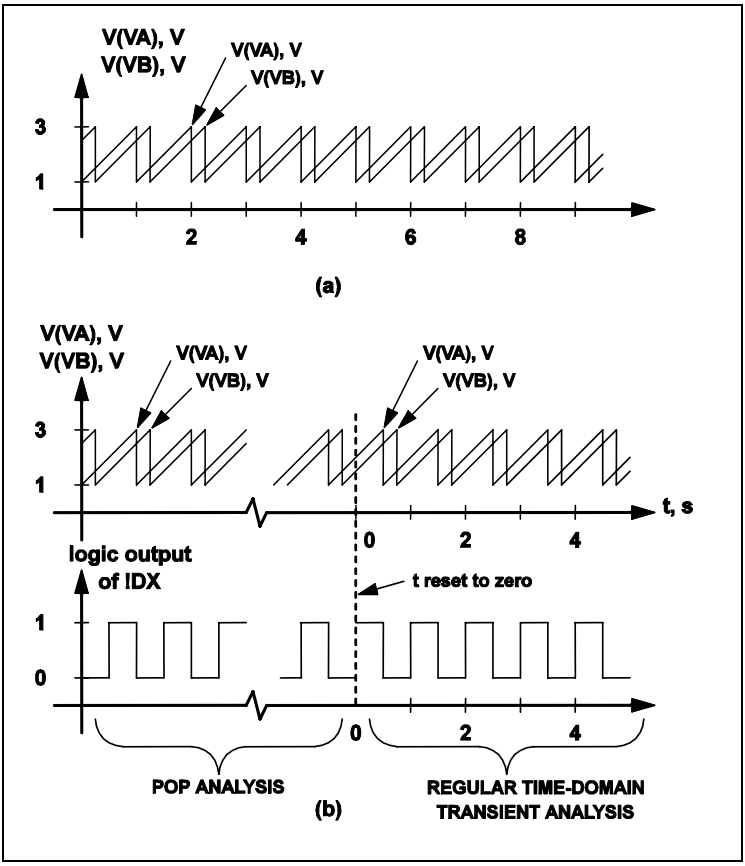
10.4 (a) Waveform of a sample piecewise-linear voltage source when no periodic operating point analysis is specified, and (b) Waveform of the same voltage source when a periodic operating point analysis is carried out

Periodic Sources

Periodic sources are left unchanged during a POP analysis, and the time variable is reset to 0.0 at the end of a POP analysis. However, it is not always true that the source value of a periodic source at the end of a POP analysis will be equal to the value of the same source at t = 0 as defined in the input file. The reason for this can be seen by examining the following statements in the input file:

```
VA 999 0 SAW V1=1 V2=3 FREQ=1K DELAY=0
+ OFF_UNTIL_DELAY=NO
VB 998 0 SAW V1=1 V2=3 FREQ=1K DELAY=250U
+ OFF_UNTIL_DELAY=NO
!DX 555 0 999 997 MX IC=0
.MODEL MX COMP RIN=10MEG ROUT=8 HYSTWD=2M
+ VOL=0 VOH=5
VDC 997 0 DC 1.999
.POP TRIG_GATE=!DX TRIG_COND=0_TO_1 MAX_PERIOD=400U
```

If no periodic operating point analysis is carried out, the source values V(VA) and V(VB) would be as shown in fig 10.5 (a). From this diagram, the source values V(VA) and V(VB) are obviously equal to 1V and 2.5 V, respectively, at t = 0. However, at the exit of a successful POP analysis, the source value of VA will be found to be equal to 2V rather than 1V. The cause of this discrepancy comes from the arbitrary reset of the time variable to zero at the end of the POP analysis. To understand this reasoning, the reader is referred to fig 10.5 (b), which shows the waveforms of V(VA), V(VB), and the logic output of the gate !DX. From the input file statements above and from fig 10.5 (b), the start of a switching cycle is seen to occur at time instants when V(VA) is rising and equal to 2V. Since each of the invisible transient analyses carried out in the periodic operating point analysis is terminated at the start of a new switching cycle, the source value V(VA) must be at 2V at the end of the POP analysis. Similarly, the source value V(VB) is equal to 1.5 V, rather than 2.5V, at the end of the POP analysis.



10.5 (a) Waveforms of two periodic sawtooth voltage sources VA and VB when no periodic operating point analysis is carried out, and (b) Waveform of the same voltage source when a periodic operating point analysis is carried out. The start of a switching cycle is defined as occurring when the output of the logic gate !DX is making a 0 to 1 transition

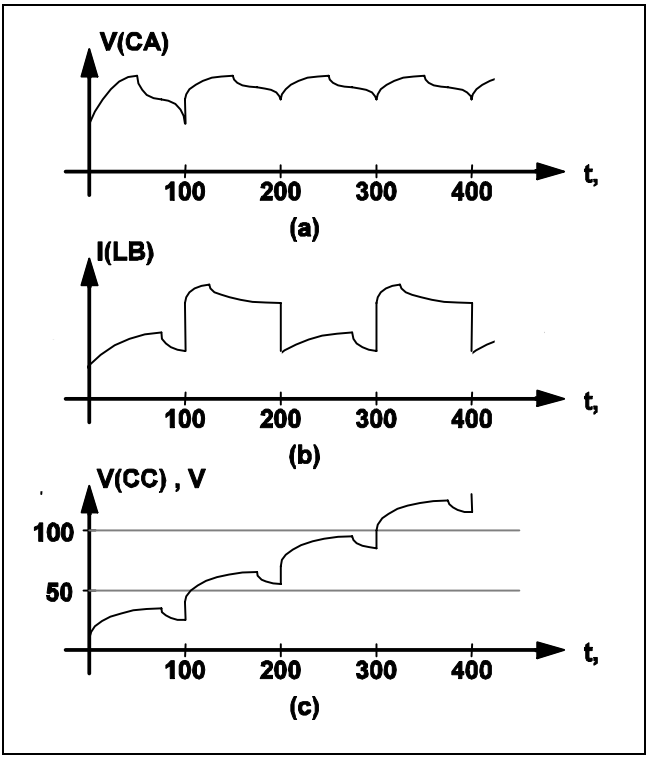
The POP_SHOWDATA option for POP Analysis

Under normal application of the periodic operating point analysis, the POP_SHOWDATA option should be turned off. If a POP analysis fails, the user can run a regular time-domain transient analysis, without the POP analysis, to make sure that the system has been correctly defined, modeled, and entered in the input file. Typographical errors in entering the initial conditions for some devices or mistakes in the definition of the switching logic can often be revealed through such a regular time-domain transient analysis.

After debugging the defined system to make sure initial conditions are within reasonable ranges and the switching logic is properly set up, the user can apply the POP analysis again to compute the steady-state solution. If the POP analysis fails again, the user may want to examine the progress of the POP analysis by repeating the POP analysis with the POP_SHOWDATA option turned on. When the POP_SHOWDATA option is turned on, a print/plot file named "XXXX.t4", where "XXXX" is the name of the input file, is generated during the POP analysis. This data file contains data for all of the print variables versus the time variable for the sequence of invisible transient analyses carried out by the POP analysis.

Due to the prediction phases interspersed between successive transient analyses, the time variable, as pointed out in See The Value of the Time Variable During the Periodic Operating Point Analysis , has no real physical meaning and significance because the POP analysis is not following any true transient experienced by the system. To make it easier to review the print variables in this data file, the time variable in this data file is artificially set so that 1) it appears to be continuous and monotonically increasing, and 2) the time instant at the end of each transient analysis carried out by the POP analysis also appears to be the instant at the start of the next transient analysis. Since the prediction phase carried out between two successive transient analyses adjusts the capacitor voltages and inductor currents so as to predict the steady-state solution, jump discontinuities in the print variables are expected in this data file at the boundary between successive switching cycles. As a result, it is not uncommon to see discontinuities in the capacitor voltages and inductor currents if a plot is made from this data file.

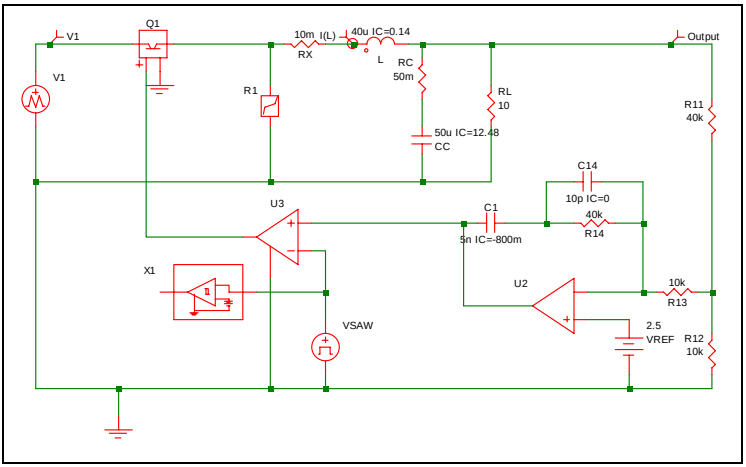
A typical waveform plotted from the "XXXX.t4" data file for a successful POP analysis is shown in fig 10.6 (a). The switching frequency for this example is 10 kHz and discontinuities in the waveform are observed at intervals of 100 microseconds. Typical waveforms plotted from the "XXXX.t4" data file for unsuccessful POP analyses are shown in fig 10.6 (b) and fig 10.6 (c). The waveform in fig 10.6 (b) is representative of a high-gain closed-loop regulated system where a small change in the capacitor voltages or inductor currents can cause a dramatic change in the mode of operation of the system. In such a case, the user may want to run a regular time-domain transient analysis until the system settles to a stable mode of operation, read off the capacitor voltages and inductor currents, and then run a POP analysis with the new initial conditions. The waveform in fig 10.6 (c) is representative of a system where the initial conditions for a few devices as supplied by the user in the input file are very far away from the values that they should be during steady-state operation. For instance, the waveform in fig 10.6 (c) suggests that, to approach steady-state operation, the initial condition associated with the capacitor CC should be increased to at least 100 V. Although the example waveforms shown in fig 10.6 are either capacitor voltages or inductor currents, all voltage and current variables specified in the .PRINT statement are generated in this data file for the user to examine the progress of the POP analysis.



10.6 Typical waveforms obtained using the POP_SHOWDATA option:(a) Successful POP analysis, waveforms settle in a periodic steady state, (b) Unsuccessful POP analysis, waveforms fluctuate with large variations from one cycle to the next, and (c) Unsuccessful POP analysis, waveforms approach unilaterally, but do not settle in a periodic steady state.

Example of Applying the POP Analysis Tool

The regulated converter in Example 5 of Chapter 9 is used here as an example in applying the periodic operating point analysis tool. The schematic associated with this example is repeated here in fig 10.7 . In particular, we would like to examine the transient response of this regulated converter when the input voltage VI is abruptly changed from 40V to 30V, with the load RL fixed at 10 . The variables of interest are the output voltage V(RL) and the current I(L) through the filter inductor. The input file describing this analysis is shown in fig 10.8 . To make the transient response more pronounced, some of the component values have been changed from those listed in Example 5 of Chapter 9. In particular, the values on C1, C14, and R14 have been changed to 0.005 F, 10 pF, and 40 k , respectively.



10.7 Regulated Converter

```
*pop-example.sxsch
.PRINT  ALL
.OPTIONS  PSP_NPT=2001  POP_ITRMAX=40
.POP  TRIG_GATE=X1.!D_CYCLE  TRIG_COND=1_TO_0  MAX_PERIOD=50u
.TRAN 2m 0
X$U2 11 14 10 opamp
VSAW 12 0 SAW V1=0 V2=5 FREQ=100k DELAY=0 OFF_UNTIL_DELAY=NO
X$U3 6 0 11 12 SIMPLIS_COMP$1
V1 2 0 PWL NSEG=3 X0=0 Y0=40 X1=100U Y1=40 X2=100U Y2=30
+ X3=100M Y3=30
VREF 14 0 2.5
L 4 5 40u IC=0.14
R12 0 8 10k
X1 12 13 PERIODIC_OP$2
R13 8 10 10k
RC 7 5 50m
RL 5 0 10
R11 8 5 40k
C1 11 9 5n IC=-800m
R14 10 9 40k
C14 9 10 10p IC=0
CC 7 0 50u IC=12.48
Q1 2 3 6 0 Q1$TP_VCQ IC=OPEN
.MODEL Q1$TP_VCQ VCQPOS VSAT=700m RSAT=100m ROFF=10Meg GAIN=10
+ TH=2.5 HYSTWD=1u LOGIC=POS LEVEL=1
!R$R1 0 3 R1$TP_SSPWLR IC=1
.MODEL R1$TP_SSPWLR VPWLR NSEG=2 X0=0 Y0=0 X1=0.7 Y1=10U
+ X2=0.8 Y2=1.00001
RX 4 3 10m
.SUBCKT PERIODIC_OP$2  1 3
.NODE_MAP IN 1
.NODE_MAP OUT 3
!D_CYCLE 3 0 1 302 M1M IC=0
VREF 302 0 DC 2.5
.MODEL M1M COMP RIN=10MEG ROUT=50 VOL=0 VOH=5
+ HYSTWD=0.001 DELAY=0
.ENDS PERIODIC_OP$2
.SUBCKT SIMPLIS_COMP$1  201 100 101 102
!DCOMP 201 100 101 102 MCOMP IC=1
.MODEL MCOMP COMP RIN=1e+007 ROUT=50 VOL=0 VOH=5
+ HYSTWD=0.001 DELAY=0
.ENDS SIMPLIS_COMP$1
.SUBCKT opamp  2 3 1
.NODE_MAP VINN 1
.NODE_MAP VINP 3
.NODE_MAP VOUT 2
RIN 3 1 5Meg
EOP 2 0 3 1 1Meg
.ENDS opamp
.END
```

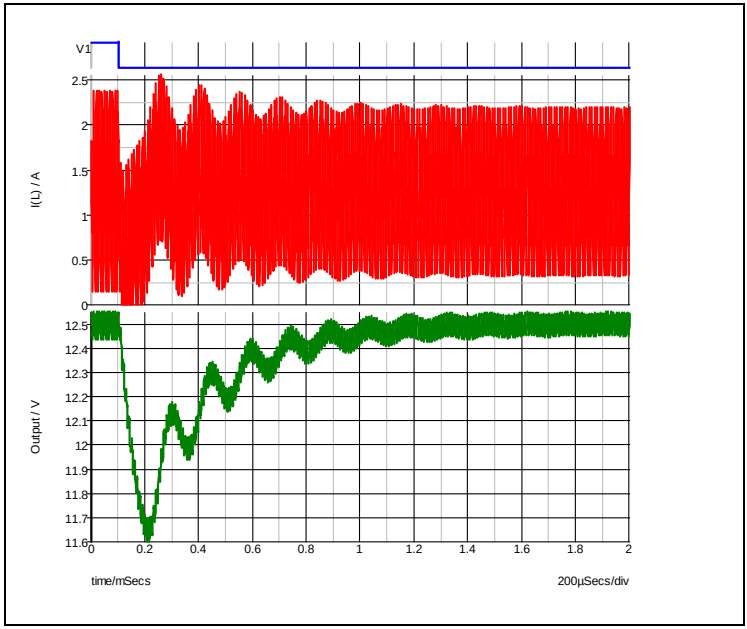
10.8 Input File

To obtain the transient response in this study, a time-domain transient simulation with the voltage of VI set at 30V is carried out after a periodic operating point analysis is applied to the system with the voltage of VI set at 40V. Notice that the initial conditions defined for the various components from Example 5 in Chapter 9 correspond to the steady-state operating conditions when the voltage of VI is at 40V. Obviously, if one already knows the steady-state solution, there is no point in running a

POP analysis to find the steady-state solution. For the purpose of illustration, the initial conditions in the input file shown in Figure 10-8 have been changed, providing SIMPLIS with initial information which is not within the immediate vicinity of the steady-state solution.

The input voltage source VI is now modeled as a piecewise-linear source instead of a DC source. Its source value jumps from 40V to 30V when t = 100 s. To find the steady-state solution of the system when the voltage of VI is 40V, the periodic operating analysis tool is invoked by using the .POP statement as shown. The comparator !D$U3 defines the start of a switching cycle as the moment when the value of the sawtooth source VSAW is decreasing and reaching the value of approximately 2.5V. Since the source VI is a piecewise-linear source, its source value is held constant at 40V, its initial value, during the POP analysis. As a result, the steady-state solution as computed by the periodic operating analysis tool corresponds to the steady-state solution of the system when VI is at 40V.

After the POP analysis is finished, SIMPLIS carries out a regular time-domain transient simulation according to the .TRAN statement for 2000s. As explained in "How POP Deals with Time Varying Sources" on page 146, the source value of VI will be at 40V for the first 100 s in this transient simulation and at 30V for the rest of the simulation. Waveforms as obtained in this time-domain transient analysis for V(VI), V(RL), and I(L) are shown in fig 10.9 .



10.9 Waveforms obtained in time-domain analysis for V(VI), V(RL), and I(L).

Chapter 11 Simplis-FX

Overview

A small-signal is applied to a system to investigate the small-signal behavior around its equilibrium. Small-signal analysis is often carried out in the frequency-domain because the resulting data can be succinctly displayed in various graphical forms, such as semi-log Bode plots and direct or inverse polar plots. Various characteristics, such as impedances, transfer functions, and stability information, can be determined from these graphical plots.

The application of small-signal frequency-domain analysis to switching piecewise-linear systems presents tremendous challenges. A Laplace transform or Laplace-transformed equivalent-circuit analysis, which is normally applied to a non-switching system to extract the small-signal frequency-domain characteristics, cannot be easily applied to a switching piecewise-linear system due to the inherent switching actions. SIMPLIS-FX is a small-signal frequency-domain analyzer specifically designed for the analysis of switching piecewise-linear systems. The analysis is based on the time-domain simulation of the switching piecewise-linear systems without having to resort to any circuit averaging or derivation of equivalent non-switching models. Instead of removing the switching actions to derive the small-signal frequency-domain characteristics, SIMPLIS-FX includes the switching action in its calculation of the small-signal frequency-domain characteristics.

While the computational aspects of SIMPLIS-FX may be daunting and tedious, the mathematical basis from which it is formulated is very simple. First, SIMPLIS-POP is used to compute the periodic operating point/trajectory of a switching piecewise-linear system. This operating point/trajectory represents the large-signal equilibrium of the system. SIMPLIS-FX can then be applied to study the small-signal behavior of the system around the large-signal equilibrium. The small-signal frequency-domain analysis is actually a sequence of analyses at discrete analysis frequencies. At each analysis frequency, the procedure of the analysis can be summarized as follows:

1. Apply small-signal stimuli in the form of voltage/current sources to the system under study. These small-signal stimuli are called the small-signal AC sources and their waveforms are time-domain sinusoidal. At each analysis frequency, the frequencies of all small-signal AC sources are set to the same value, the analysis frequency, and their amplitudes are set to infinitesimally small values. Since the frequency of all small-signal AC sources are set to the analysis frequency, the analysis frequency is also frequently called the excitation frequency.
2. The equilibrium of the system under perturbation from the small-signal is computed next. This new equilibrium, although at an infinitesimally small distance away from the large-signal equilibrium computed by the periodic operating-point (POP) analysis, is definitely not the same as the large-signal equilibrium. While the large-signal equilibrium is periodic with a frequency equal to the periodic operating frequency, or the switching frequency, as determined by the POP analysis, this new equilibrium is periodic with a frequency that is equal to the highest common factor between the analysis frequency and the periodic operating frequency.

3. Fourier analysis is then applied to the new equilibrium to extract the small-signal response of the system at the analysis frequency.

Since SIMPLIS-FX is based on time domain simulations, it can handle a switching piecewise-linear system with any structure (topology), any mode of operation, and any control scheme, and under fixed or variable switching frequency as long as the following two conditions are satisfied:

1. The system can be simulated in the time domain via SIMPLIS-TX, and
2. SIMPLIS-POP is able to successfully compute the periodic operating point/trajectory of the system.

For example, multiple-switch multiple-output, multiple feedback loop converters are easily handled by SIMPLIS-FX because the analysis is constructed for general switching piecewise-linear systems without any assumption or restriction placed on the number of switches, outputs, or feedback loops.

Some switching power supplies are designed to have very high DC gain to improve the line/load regulation of the output voltage(s). The measurement of the loop gain of these systems cannot be carried out with the loop opened because the high DC gain of the system may drive the opened-loop system to operate at a vastly different operating equilibrium from the original closed-loop equilibrium. Thus, the ability to evaluate the loop-gain of a switching power supply with the feedback loop closed is very useful. The effect of the parasitic elements on the frequency response is usually minimal. If it is suspected that a few parasitic elements are significantly affecting the frequency response of the system, SIMPLIS-FX can be relied upon to verify such a hypothesis. SIMPLIS-FX can accurately predict the impact that the parasitic elements might have on the frequency response because SIMPLIS-FX does not assume the system variables to be slowly varying within one switching period and does not remove the switching actions during the process of deriving the small-signal response.

The algorithm behind SIMPLIS-FX is rigorously derived, making it accurate and robust. For example, the analysis frequency is not limited to less than half of the switching frequency. Undeniably, aliasing is going to be present when a switching system is excited at a frequency over half of the switching frequency. In such a situation, SIMPLIS-FX can accurately compute the response of the system at the excited frequency, whether it is below or above the switching frequency. Theoretically, the small-signal frequency-analysis algorithm behind SIMPLIS-FX is accurate to within 0.5 dB and 1 degree at each analysis frequency from DC to infinity. Practically, the accuracy of the analysis and the highest analysis frequency that can be applied and still maintain a prescribed accuracy depend on how accurately the physical components are modeled in the switching piecewise-linear system. If there is a noticeable discrepancy between the measured frequency response in the laboratory and the data generated by SIMPLIS-FX, you can trust SIMPLIS-FX and concentrate on

1. determining that the simulated system represents, with reasonable accuracy, the system measured in the laboratory,
2. improving the device models of any components that you believe have not been adequately modeled, and
3. checking the laboratory measurement setup to make sure that the measurements are valid, since laboratory measurements of switching systems with small-signal excitation are inherently noisy and noise can easily lead to measurement errors.

In summary, the features of SIMPLIS-FX are as follows:

1. It is based on time-domain simulation via SIMPLIS-TX.
2. It relies on SIMPLIS-POP to compute the large-signal periodic operating equilibrium of the system.
3. It is general and versatile:

   it handles any pulse-width modulated (PWM) circuit topologies such as boost, buck, buck-boost, Cuk, SEPIC, half-bridge, full-bridge, etc.,

   it handles any resonant circuit topologies such as series resonant, parallel resonant, quasi resonant, phase-shifted resonant, etc.,

   it handles any mode of operation such as continuous-mmf (CMM) mode, discontinuous-mmf (DMM) mode, etc.,

   it handles any control scheme such as voltage-mode control, peak-current-mode control, average-current-mode current, charge control, etc.,

   it handles both fixed-frequency as well as variable-frequency systems, and

   it easily handles multiple-switch multiple-output converters

   it can evaluate the loop-gain of a system while the loop is closed

   it can evaluate the effect of the parasitic elements on the frequency response,

   it is accurate for analysis frequencies above the switching frequency, and

   it is accurate to within 0.5 dB and 1 degree

Statements Relating AC Analysis

The small-signal frequency-domain analysis is a sequence of individual analyses at a set of discrete excitation frequencies. Since the excitation frequency is monotonically increased from the starting frequency to the stopping frequency, this sequence of analyses is usually called the "swept AC analysis", or "AC analysis with swept frequencies". The excitation frequency of the first analysis is set to "start_freq". In each subsequent analysis, the excitation frequency is increased according to the "sweep_type" and "n_pt" parameters. The sequence of analyses is stopped when the excitation frequency exceeds the parameter value of "stop_freq". At each of the excitation or analysis frequencies, periodic small-signal stimuli having the same frequency as the analysis frequency are applied to excite the system around its operating point or equilibrium. The response of the system at this analysis frequency is then measured. The analysis frequencies are defined through an analysis statement and the small-signal stimuli are defined through device statements. In addition, you have the option of analyzing the system in either the continuous domain or the discrete domain by using an option statement.

.AC Analysis Statement

The .AC analysis statement instructs SIMPLIS to carry out the SIMPLIS-FX Small-Signal Frequency-Domain analysis. The format of the .AC statement is:

.AC *sweep_type n_pt start_freq stop_freq*

where

.AC	is the three-character keyword ".AC", standing for small-signal frequency-domain analysis.
sweep_type	is either the three-character keyword "DEC", the three-character keyword "OCT", or the three-character keyword "LIN".
n_pt	is a positive integer. If "sweep_type" is set to "DEC", the frequency of analysis will be swept in a logarithmic manner and n_pt represents the number of points per decade in the swept frequency. If "sweep_type" is set to "OCT", the frequency of analysis will be swept in a logarithmic manner and n_pt represents the number of points per octave in the swept frequency. If "sweep_type" is set to "LIN", the frequency of analysis is swept in a linear manner and n_pt represents the total number of points in the linear frequency sweep. If "sweep_type" is set to be "LIN", n_pt must be an integer larger than one.
start_freq	is a positive floating-point number representing the starting frequency of the sweep.
stop_freq	is a positive floating-point number representing the stopping frequency of the sweep.

The .AC statement can be specified in any one of the three formats shown. Just like any other analysis statement, the .AC statement can only appear within the scope of definition of the main circuit. There can be no more than one .AC statement in an input file. Since the SIMPLIS-FX Small-Signal Frequency-Domain analysis is specifically designed for the small-signal analysis of switching piecewise-linear systems around its periodic operating point, the periodic operating-point (POP) analysis must be carried out before the small-signal analysis can be applied. Therefore, the .POP statement must appear before the .AC statement in the input file. If both the time-domain transient analysis and the small-signal frequency-domain analysis are specified in an input file, the .AC analysis statement for the small-signal frequency-domain analysis must appear before the .TRAN analysis statement for the time-domain transient analysis.

Option Statement Associated with AC Analysis

There is one option statement associated with the SIMPLIS-FX small-signal frequency-domain analysis. The format of this option statement is:

.OPTIONS FREQ_DOMAIN=*D*

where

.OPTIONS	is the eight-character keyword ".OPTIONS" indicating this is an option statement.
FREQ_DOMAIN=	is the 12-character keyword "FREQ_DOMAIN="
D	is either the character 'S' indicating the continuous domain or the character 'Z' indicating the discrete domain.

By default, a frequency-domain analysis is carried out in the continuous domain. The statement:

```
.OPTIONS FREQ_DOMAIN=Z
```

will override the default and instruct SIMPLIS-FX to analyze the system using discrete domain techniques. If the continuous domain is chosen, the waveform of each small-signal stimulus will be a continuous sinusoidal function of the time variable. If the discrete domain is chosen, the waveform of each small-signal stimulus as a function of the time variable is equal to the result of applying the sample and hold to a continuous sinusoidal function of the time variable. See “Synopsis of Small-Signal AC Analysis” on page 160 for a more detailed explanation of the difference between using the different domains in a small-signal frequency-domain signal analysis.

Statement Defining a Small-Signal AC Source

A small-signal stimulus is defined via a small-signal AC voltage or current source. The formats for defining small-signal AC voltage and current sources are:

V*name*> *n*+ *n*- AC *amplitude phase*

and

I*name* *n*+ *n*- AC *amplitude phase*

where

V	is the one-character element keyword "V" indicating a voltage source.
I	is the one-character element keyword "I" indicating a current source.
name	is the individual name of the device.
n+	is the name of the positive node, and is a nonnegative integer.
n-	is the name of the negative node, and is a nonnegative integer.
AC	is the two-character keyword "AC" to signify that this is a small-signal AC source.
amplitude	is a positive floating-point number representing the amplitude of this small-signal AC source relative to any other specified small-signal AC sources. The specification of the amplitude parameter is optional. A default value of 1.0 (unit) will be used if no value is specified.
phase	is a floating-point number representing the relative phase of this small-signal AC source in degree. The specification of phase is optional. If the phase parameter is to be specified for a certain source, the amplitude parameter must also be specified. The default for phase is 0.0 degree.

Since device statements can appear within the scope of definition of any general circuit, the small-signal AC voltage/current sources can be defined in the main circuit

as well as in any subcircuit. For a more detailed explanation of the "amplitude" and the "phase" parameters, see “Synopsis of Small-Signal AC Analysis” on page 160.

Synopsis of Small-Signal AC Analysis

After the periodic-operating point/trajectory of the system has been determined by the periodic-operating point (POP) analysis, the SIMPLIS-FX Small-Signal Frequency-Domain Analysis can then be applied to compute the small-signal response of the system in a small neighborhood around the large-signal periodic equilibrium. For illustration purposes, let us assume that the following .AC analysis statement was specified in the input file:

```
.AC DEC 10 250 25K
```

This analysis statement instructs SIMPLIS-FX to carry out the small-signal frequency-domain analysis by sweeping the excitation frequency of all small-signal AC sources in a logarithmic manner from 250 Hz to 25 kHz. The number of analysis frequencies per decade is set to 10. You can easily verify that this analysis statement leads to the following excitation frequencies:

```
250.000 Hz 314.731 Hz 396.223 Hz
. . .
1.57739 kHz 1.98582 kHz 2.50000 kHz
3.14731 kHz 3.96223 kHz
. . .
15.7739 kHz 19.8582 kHz 25.0000 kHz
```

As the sequence of small-signal analyses at these excitation frequencies is carried out, SIMPLIS-FX reports the progress by printing a matrix of numbers which represent the "percent complete" of the swept-frequency analysis. For example, a screen display such as:

```
SMALL-SIGNAL FREQUENCY-DOMAIN ANALYSIS
CONTINUOUS S-DOMAIN

01 02 03 04 05 06 07 08 09 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50
51 52
Elapsed time : 0 hr 0 min 6 sec
CPU time : 0 hr 0 min 4.64 sec
Analyzed freq. : 2.500000000000e+03 Hz
```

means that 52 percent of the small-signal frequency-domain analyses have been completed and that the excitation frequency has been swept past 2.5 kHz.

The "percent complete" shown in the small-signal frequency-domain analysis is calculated based on the number of expected analyses at discrete excitation frequencies and not based on the CPU time. Since a logarithm sweep of 10 points per decade from 250 Hz to 25 kHz is specified in the preceding example, there will be a total of 21 analyses starting from 250 Hz and stopping at 25 kHz. The analysis at 2.5 kHz is the 11th one in this sequence of analyses. Hence, if SIMPLIS-FX has just finished the analysis at 2.5 kHz, the percent complete is reported as (11 / 21) x 100 = 52%.

The actual CPU time involved in each individual analysis varies. As a rule of thumb, the CPU time is linearly proportional to the excitation frequency. When the excitation frequency is 1 decade or more below the periodic operating frequency computed in the POP analysis, SIMPLIS-FX is extremely fast. When the excitation frequency is close to the periodic operating frequency computed in the POP analysis, SIMPLIS-FX is reasonable in computation speed. When the excitation frequency is 1 decade or more above the periodic operating frequency computed in the POP analysis, the computation time taken by SIMPLIS-FX will be much longer, since SIMPLIS-FX must determine the high excitation frequency affect on the system response.

This brief synopsis is adequate as a reference for using the SIMPLIS-FX Small-Signal Frequency-Domain Analyzer. It is recommended that any first-time user, as well as those interested in more details of SIMPLIS-FX, read the following subsections of this Section to better understand the behavior of small-signal AC sources and various large-signal time-varying sources during the small-signal frequency-domain analysis. “Behaviour of AC Sources in Transient and POP” on page 166 describes how the small-signal AC sources are treated in both the time-domain transient analysis and the periodic operating-point analysis.

Amplitude of Small-Signal AC Sources

If the continuous domain is chosen for the small-signal frequency-domain analysis, all of the small-signal AC sources are sinusoids. If the discrete domain is chosen instead, the waveforms of all of the small-signal AC sources are the result of applying a sample-and-hold process to a sinusoid. In both cases the original sinusoidal waveform is completely defined by three parameters: the frequency, the amplitude, and the phase of the sinusoidal waveform. The frequency of each small-signal AC source is set to the analysis frequency defined by the "sweep_type" and "n_pt" parameters in the .AC analysis statement. The "amplitude" parameter is explained in this subsection.

The "amplitude" parameter in the device statement defining a small-signal AC source should be regarded as a relative quantity rather than an absolute quantity. Suppose the following statements appear in the input file:

```
V1 1 0 AC 2 -45 V2 4 0 AC I3 7 9 AC 5 60
```

Since the amplitude and the phase parameters of V2 are not specified, the defaults of 1.0 unit and 0.0 degree are used. Although the amplitude parameter of V1 is set to 2, the actual amplitude of V1 during the small-signal frequency-domain analysis is not equal to 2.0 V. By definition, a small-signal analysis is the examination of the behavior of a system around its equilibrium when it is perturbed from its equilibrium with stimuli of infinitesimally small amplitude. The amplitude of the three sources V1, V2, and I3 are all set to infinitesimally small values by SIMPLIS-FX during the small-signal analysis. However, the ratios between the amplitude of these three sources are maintained according to the amplitude parameters in the statements defining the small-signal AC sources. In this example, the amplitude of V1 is always set to twice the amplitude of V2 and the amplitude of I3 is always set to five times as large as the amplitude of V2. Another way to interpret the three device statements above is that the amplitude of V1, V2, and I3 are 2, 1, and 5 infinitesimally small units, respectively.

The excitation of a nonlinear system by sinusoidal inputs with finite amplitudes generates responses not only at the excitation frequency, but also at harmonics or subharmonics of the excitation frequency. For a true small-signal analysis, the amplitude of the sinusoidal inputs must be made infinitesimally small so the responses

at the higher harmonics and/or the subharmonics are insignificant compared to the response at the excitation frequency. This is equivalent to saying that the response of a nonlinear system appears linear when the amplitude of the excitation sources are sufficiently small. The same can be said about switching piecewise-linear systems.

Making small-signal measurements on a breadboard system in the laboratory presents some practical challenges in choosing the amplitude of the exciting sinusoids. Ideally, we would like the amplitude of the exciting sinusoids to be as small as possible to avoid the nonlinear effects, but the amplitude of these exciting sinusoids cannot be too small. If these amplitudes are too small, the signal-to-noise ratio would be low and it will be extremely difficult to get an accurate measurement. This is especially true for switching systems which inherently carry large-signal noises at the switching frequency and higher harmonics. SIMPLIS-FX uses a proprietary algorithm to make sure that the amplitude of all small-signal AC sources are infinitesimally small so as to generate linear responses, and the infinitesimally small responses are accurately resolved in the presence of the large-signal switching noises.

Phase Delay of Small-Signal AC Sources

Similar to the "amplitude" parameter, the "phase" parameter in the device statements defining a small-signal AC source is a relative quantity rather than an absolute quantity. Suppose again that the following device statements appear in the input file:

```
V1 1 0 AC 2 -45 V2 4 0 AC I3 7 9 AC 5 60
```

The amplitude and the phase parameters of V2 are not specified, so the default values of 1.0 unit and 0.0 degree are used. The phase parameter is a relative quantity. In this example, the phase of I3 is set to 60 degrees ahead of the phase of V2 at any excitation frequency, and the phase of V1 is set to 45 degrees behind the phase of V2.

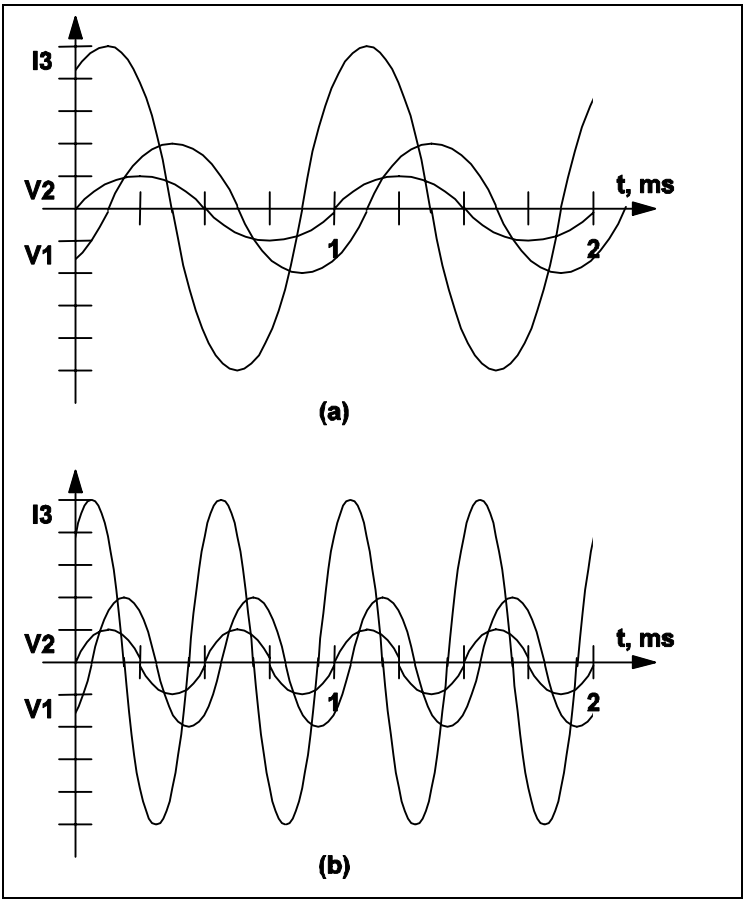
You will rarely need more than one small-signal AC source with different phase delay in the small-signal frequency-domain analysis of switching piecewise-linear systems. The phase parameter has been provided for backward compatibility with existing circuit simulators such as SPICE.

Sample Waveforms of AC Sources Continuous Domain

The waveform of a small-signal AC source is sinusoidal if the continuous domain is used for the small-signal frequency-domain analysis. For example, the following device statements specify three small-signal AC sources V1, V2, and I3 with amplitudes equal to 2, 1, and 5 units, respectively.

```
V1 1 0 AC 2 -45 V2 4 0 AC I3 7 9 AC 5 60
```

The waveforms of these three sources at an analysis frequency of 1 kHz are displayed in 11.1(a). The waveforms of these sources at an analysis frequency of 2 kHz are displayed in 11.1(b). Since the phase parameter is relative, these waveforms have been arbitrarily drawn with V2 having a positive-slope zero crossing at t=0. Once the phase of V2 has been arbitrarily chosen, the phases of V1 and I3 are fixed at 45 degrees behind and 60 degrees ahead of the phase of V2, respectively.



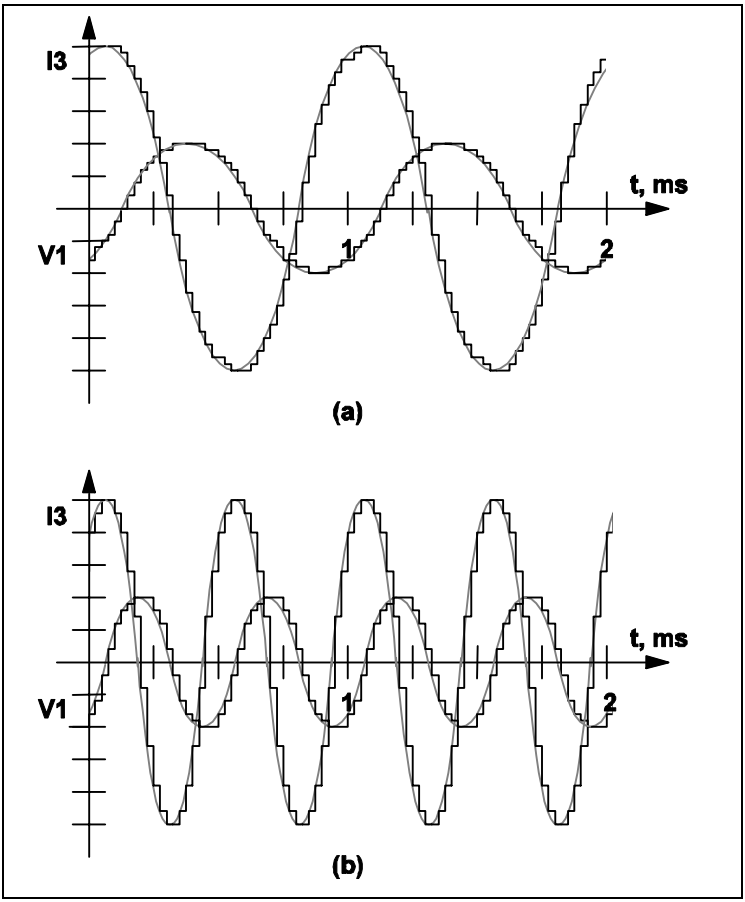
11.1 Waveforms for the small-signal AC source examples V1, V2, and I3

Sample Waveforms of AC Sources Discrete Domain

If the discrete domain is used, the waveform of a small-signal AC source as a function of time is the result of applying an ideal sample-and-hold process to a sinusoidal waveform. This ideal sampling is taken at the beginning of a switching period/cycle where the beginning of a switching period/cycle is defined through the .POP analysis statement. The value of a small-signal AC source is held constant for the remaining of the switching period/cycle until another set of samples is taken at the start of the next switching period/cycle. The sample device statements shown in “Sample Waveforms of AC Sources Continuous Domain” on page 162 are repeated here for illustration:

```
V1 1 0 AC 2 -45 V2 4 0 AC I3 7 9 AC 5 60
```

Suppose the analysis frequency is 1 kHz and the switching frequency, or the periodic operating frequency, computed by the POP analysis is 50 kHz. Fig. 11-2(a) shows the sinusoidal waveforms associated with V1 and I3 in dash lines and their actual waveforms in solid lines. It can be seen that the waveforms of V1 and I3 are obtained by applying sample-and-hold to the corresponding sinusoidal waveforms. The associated waveforms of V2 are expressly omitted in fig 11.2 (a) to reduce the cluttering of the figure. The waveforms associated with V1 and I3 should sufficiently illustrate the sample-and-hold process. Fig 11.2(b) show similar waveforms of V1 and I3 when the analysis frequency is at 2 kHz.



11.2 Waveforms for the small-signal AC source examples, V1 and I3

Continuous and Discrete Domain Differences

The most obvious difference between using the continuous domain or the discrete domain in the small-signal analysis is the source values of the small-signal AC sources. As illustrated in the previous subsections, the source value of a small-signal AC source is a continuous sinusoidal waveform if the continuous domain is used. If the discrete domain is used, the waveform of a small-signal AC source is the piecewise-constant waveform resulting from the application of a sample-and-hold to a related continuous sinusoidal waveform.

Another difference between using the continuous domain and the discrete domain in the small-signal analysis is the way the responses are analyzed. In the case of the continuous domain, Fourier analysis is directly applied to any response of the system to extract the harmonic with the same frequency as the analysis frequency. In the case of the discrete domain, a sample-and-hold process is applied to each response of the system, and Fourier analysis is then applied to the result of this sample-and-hold process. The sampling is set to occur, just like the small-signal AC sources, at the beginning of a switching cycle where this beginning is defined through the .POP analysis statement.

Whether continuous domain or discrete domain is used, the effects of the implicit sample-and-hold that occurs in a switching system are taken into consideration by SIMPLIS-FX. For example, if a simple switch, S, is controlled by a control signal, cs(t), an effective sample-and-hold process of cs(t) is taken place at every moment switch S switches position.

Another example of implicit sample-and-hold occurs at the two inputs of a comparator. Whenever the comparator switches its output logic state, we have, in effect, a sample-and-hold of the two analog inputs of the comparator. If you are familiar with the peak-current control of energy-storage dc-to-dc converters, recall that peak-current control would approach unstable operation as the duty ratio of the power switch approaches 50%, provided no compensation ramp is applied to the control. Using existing modeling methods, such instability can only be predicted by explicitly adding a sample-and-hold function block, or any approximation of such, from the controlling signal to the power stage. Such explicit addition is not necessary when these control strategies are analyzed with SIMPLIS-FX because the implicit sample-and-hold process that occurs in the circuit is taken into consideration in the computation of the frequency response. As a result, the schematic that is used for time-domain simulation is the schematic that will be analyzed in frequency-domain, eliminating the need to replace parts of a schematic by averaged-circuit equivalents and eliminating the need to add extraneous components or function blocks when small-signal frequency-domain analysis is performed. Using SIMPLIS-FX, you can study how a converter under peak-current control scheme approaches instability as the duty ratio approaches 50% when no compensation ramp is applied in the control and this instability can be analyzed using either the continuous domain or the discrete domain. The discrete domain is useful if you want to study a system using digital control techniques. For most applications, the use of the continuous domain is adequate.

AC Analysis Behaviour of Time-Varying Sources

Since the SIMPLIS-FX Small-Signal Frequency-Domain Analyzer is specifically designed for small-signal analysis of switching piecewise-linear systems in the near neighborhood of the periodic operating point/trajectory, large-signal time-varying sources are treated the same way in the small-signal analysis as they were treated

during the periodic operating point (POP) analysis. All periodic large-signal time-varying sources are treated as active periodic sources with time-varying source values while all aperiodic large-signal time-varying sources are treated as DC sources during the small-signal frequency-domain analysis. Section 10.3.2 explains the treatment of the time variable and the treatment of various large-signal time-varying sources during the periodic operating-point (POP) analysis. The key features of these treatments are summarized here for convenience:

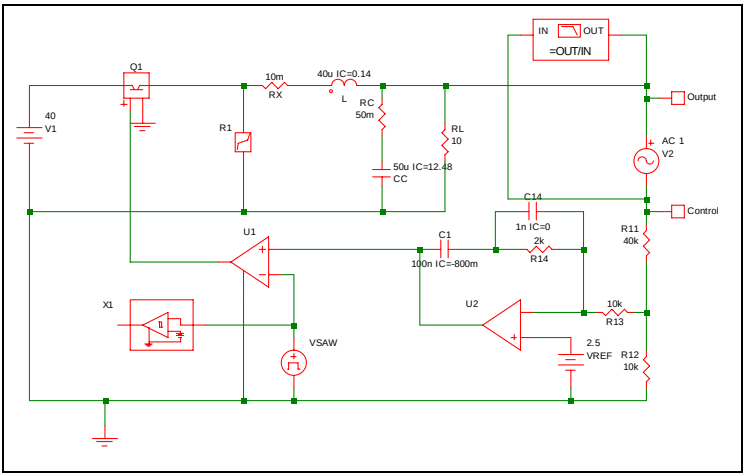
1. Simplis resets the time variable t to zero at the end of a POP analysis, i.e. at the end of the circuit's period as defined by the circuit and the POP trigger gate. Simplis-FX starts its simulations for each excitation frequency at the same t=0 point;
2. during the small-signal frequency-domain analysis, aperiodic sources are changed to DC sources. The value of each DC source is set equal to the calculated values of the corresponding aperiodic source at the time equivalent to the end of the circuit's period. Aperiodic sources include exponential pulse sources, piecewise-linear sources, and sinusoidal/cosinusoidal sources with non zero damping coefficients;
3. the source values of all periodic large-signal time-varying sources such as sawtooth sources, triangular sources, square-wave sources, pulse sources, and sinusoidal/cosinusoidal sources with zero damping coefficients are considered active during the small-signal frequency-domain analysis. That is, they maintain the same time-varying waveforms as if they were used in a large-signal time-domain analysis. These sources are not turned into DC sources because their time-varying waveforms are essential to the periodic operation of the switching piecewise-linear systems. Turning these sources into DC sources is tantamount to eliminating the switching actions of the system under study and the small-signal analysis would not be able to reveal the true small-signal nature of the system around its periodic equilibrium.

Behaviour of AC Sources in Transient and POP

Small-signal AC sources have no effect on any analysis other than the small-signal frequency-domain analysis. For example, a small-signal voltage source is turned into a short circuit during the time-domain transient analysis and during the periodic operating-point analysis. Similarly, a small-signal current source is turned into an open circuit during the time-domain transient analysis and during the periodic operating-point analysis. As a result, leaving small-signal AC sources in the input file has no effect on any analysis except the small-signal frequency-domain analysis. Hence it is not necessary for you to remove the definition of the small-signal AC sources from the input file in order to run other types of analyses.

Example of Applying the AC Analysis Tool

The regulated converter used in the illustration of the Periodic Operating Point Analysis in Chapter 10 is repeated here to show how the SIMPLIS-FX Small-Signal Frequency-Domain can be applied to determine the loop gain of a closed-loop regulated converter. The schematic of this converter, including the small-signal AC source, is shown in Fig. 11.3. The input file of this analysis is shown in Fig. 11.4.



11.3 Small-signal excitation to determine the loop gain of the regulated converter.

```
Example in User Manual with Small-Signal Analysis
.NODE_MAP Output 15
.NODE_MAP Control 14
.AC DEC 40 1u 50k
.PRINT  ALL
.OPTIONS  PSP_NPT=2001  POP_ITRMAX=40
.POP  TRIG_GATE=X1.!D_CYCLE  TRIG_COND=1_TO_0
+ MAX_PERIOD=50u
X$U2 9 13 8 opamp
VSAW 11 0 SAW V1=0 V2=5 FREQ=100k DELAY=0 OFF_UNTIL_DELAY=NO
X$U1 5 0 9 11 SIMPLIS_COMP$1
V1 2 0 40
V2 15 14 AC 1
VREF 13 0 2.5
L 4 15 40u IC=0.14
R12 0 10 10k
X1 11 12 PERIODIC_OP$2
R13 10 8 10k
RC 6 15 50m
RL 15 0 10
R11 10 14 40k
C1 9 7 100n IC=-800m
R14 8 7 2k
C14 7 8 1n IC=0
CC 6 0 50u IC=12.48
Q1 2 3 5 0 Q1$TP_VCQ IC=OPEN
.MODEL Q1$TP_VCQ VCQPOS VSAT=700m RSAT=100m ROFF=10Meg GAIN=10
+ TH=2.5 HYSTWD=1u LOGIC=POS LEVEL=1
!R$R1 0 3 R1$TP_SSPWLR IC=1
.MODEL R1$TP_SSPWLR VPWLR NSEG=2 X0=0 Y0=0 X1=0.7 Y1=10U
+ X2=0.8 Y2=1.00001
RX 4 3 10m
.SUBCKT PERIODIC_OP$2  1 3
.NODE_MAP IN 1
.NODE_MAP OUT 3
!D_CYCLE 3 0 1 302 M1M IC=1
VREF 302 0 DC 2.5
.MODEL M1M COMP RIN=10MEG ROUT=50 VOL=0 VOH=5
+ HYSTWD=0.001 DELAY=0
.ENDS PERIODIC_OP$2
.SUBCKT SIMPLIS_COMP$1  201 100 101 102
!DCOMP 201 100 101 102 MCOMP IC=1
.MODEL MCOMP COMP RIN=1e+007 ROUT=10 VOL=0 VOH=5
+ HYSTWD=0.1 DELAY=0
.ENDS SIMPLIS_COMP$1
.SUBCKT opamp  2 3 1
.NODE_MAP VINN 1
.NODE_MAP VINP 3
.NODE_MAP VOUT 2
RIN 3 1 5Meg
EOP 2 0 3 1 1Meg
.ENDS opamp
.END
```

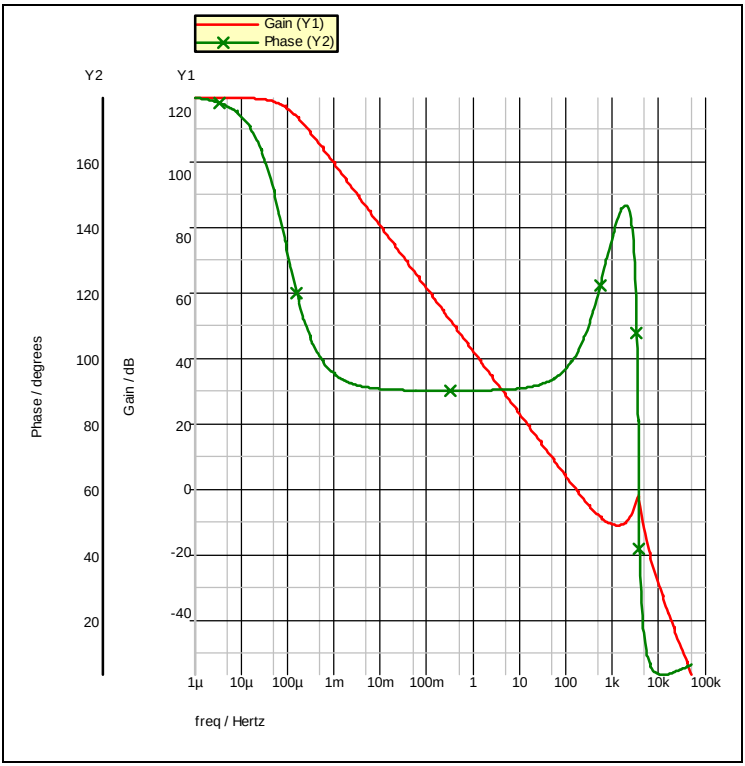
11.4 Input File (as generated by SIMetrix)

Notice that a small-signal AC source, VSS, is inserted in the feedback loop between the output and control nodes. If the impedance from the control node to ground is

much higher than the impedance from the output to ground, the loop gain G(jw) of such a regulated system is accurately approximated by the following complex ratio:

$G(j\omega) = -\ Voutput(j\omega)/Vcontrol(j\omega)$

This is the function of the "Bode Plot Probe" shown at the top right of the schematic 11.3. The result of G(jω) for this converter is shown in fig 11.5. In fig 11.5 , the gain cross-over occurs at 147 Hz, and the phase margin is about 100 degrees. Although the phase margin is high, indicating the converter is stable as it is, there is room for improvement to this design. The low cross-over frequency means the transient response of this regulated converter would be slow and would take a long time to settle back to the periodic operating equilibrium. Moreover, a zero should be introduced to the loop gain function to compensate for the pair of complex poles at 3.56 kHz which is caused by the output filter formed by L and C. Otherwise, the sharp drop in the phase due to the pair of complex poles can have a very detrimental effect on the stability of the system if the cross-over frequency is pushed higher. Changing the value of R14 will effect the gain cross-over frequency and the phase margin of this converter. As a matter of fact, this converter becomes unstable, with a phase margin of -6 degrees, when the resistor R14 in the feedback loop is raised to 16 KΩ!



11.5 Magnitude and Phase of G(jω)

Index

!C 46
!D 47
!L 46
!R 45
.AC 157
.ELSE 111
.END Statement 18
.ENDIF 111
.ENDS Statement 18, 88
.ENDWHILE 112
.IF 111
.MODEL 50
.OPTIONS 142, 158
.POP 136
.SIMULATOR 107, 110
.SUBCKT Statement 18, 88
.TRAN 104
.VAR 111
.WHILE 112

A
Aborting run 109
AC Analysis 155
    continuous 165
    discrete 165
AND Gate Model 73
ANDk 48, 51, 73
APEAK 32

C
Capacitors
    linear 23
    PWL 46
    PWL Model 55
CLK_DFF 48, 51, 80
CLK_JKFF 48, 51, 79
CLK_SRFF 48, 51, 77
CLK_TFF 48, 51, 82
Clocked Data Flip-Flop 80
Clocked J-K Flip-Flop 79

Clocked Set-Reset Flip-Flop 77
Clocked Toggle Flip-Flop 82
CLOSE 44
Comments 10, 11
COMP 48, 51, 67
Comparator Model 67
Conditional Lines 111
Continuation lines 10, 11
Control Statements 19, 95

D
DAMP_COEF 32
Data Files 113
DELAY 24, 26, 28, 30
DELAY_F 36
DELAY_R 36
Device Statements 18, 20
DRATIO 26, 30

E
ENABLE_LEVEL 83
Examples 117
Exclusive-OR Gate Model 69
EXPAND 96

F
F11 106
Fields
    max. length 16
Format
    device statements 20
FREQ 24, 26, 28, 30, 32
FREQ_DOMAIN 98, 158

G
GAIN 59
Gates
    simple logic 47

H
HYSTWD 57, 59, 65, 67, 69, 70, 72, 73, 74, 75, 77, 79, 80, 82, 83

I
IC 45, 46, 47
ICQNEG 45, 51, 59
ICQPOS 45, 51, 59
ICSW 51

IGNORE_UNITS 99
Inductors
    linear 23
    mutual 40
    PWL 46
    PWL Model 55
    saturable 129
Initial Conditions 99
Input deck 17
Input netlist 17
INV 48, 51, 65
Inverter Model 65
IPWLR 51, 54

L
LATCH 48, 51, 83
Latch 83
LEVEL 59
Library Search 110
Lines
    max. length 16
LOGIC 57, 59, 69, 70, 72, 73, 74, 75, 77, 79, 80, 82, 83
Looping 112

M
MAPNODE0 96
MAX_PERIOD 136
MAX_TOPOLOGY 97
Model Statements 18, 50
Models
    AND gate 73
    clocked J-K flip-flop 79
    clocked set-reset flip-flop 77
    clocked toggle flip-flop 82
    comparator 67
    exclusive OR 69
    inverter 65
    latch 83
    NAND gate 74
    NOR gate 72
    OR gate 70
    PWL Capacitor 55
    PWL Inductor 55
    PWL Resistor 52
    set-reset flip-flop 75

simple logic device 64
simple switch 57
simple transistor switch 59

N
N_WIND 43
Names
device 11
model 12
subcircuit 12
NAND Gate Model 74
NANDk 48, 51, 74
Netlist
adding extra lines 106
Netlist Preprocessor 109
NEW_ANALYSIS 98
Node Names 20
NOR Gate Model 72
NORk 48, 51, 72
NSEG 38, 52, 55

O
OFF_UNTIL_DELAY 24, 26, 28, 30, 32
OPEN 44
Operational Amplifier 121
Option Statements 95
OR Gate Model 70
ORk 48, 51, 70

P
Parameters 21, 111
Passing Parameters to Subcircuits 111
PDELAY 32
Periodic Operating Point 135
Piecewise Linear Resistors 45
POP 135
POP_CONVERGENCE 142
POP_ITRMAX 96, 142
POP_OUTPUT_CYCLES 97, 143
POP_SHOWDATA 97, 142, 149
POP_USE_TRAN_SNAPSHOT 96, 142
Preprocessor 109
PSP_END 96
PSP_NPT 96
PSP_START 96

PWL 38
PWLC 51
PWLL 51

R
Resistors
linear 22
PWL 45
PWL Model 52
RIN 65, 67, 69, 70, 72, 73, 74, 75, 77, 80, 82, 83
ROFF 57, 59
RON 57
ROUT 65, 67, 69, 70, 72, 73, 74, 75, 77, 79, 80, 82, 83
RSAT 59
Running SIMPLIS
aborting run 109
external netlist 107
from a script 107
from DOS prompt 108
schematic 106
RunSIMPLIS command 107

S
Saturable Inductor 129
SAW 24
SCR 131
Set-Reset Flip Flop Model 75
Simple Logic Device Models 64
Simple Logic Gates 47
Simple Switch Models 57
Simple Switches 43
Simple Transistor Switch Models 59
Simple Transistor Switches 44
Simplis Data Files 113
Simplis-FX 155
Simplis-POP 135
SNAPSHOT_INTVL 98
SNAPSHOT_NPT 98
Sources
AC 159
cosine 34
current 23
current controlled 42
exponential 36
fixed 23

independent 23
pulse 30
PWL 38
sawtooth 24
sine 32
square 28
triangular 26
voltage 23
voltage controlled 41
SRFF 48, 51, 75
Subcircuit Definition 86
Subcircuits 18
calls 48, 92
definition 86
external nodes 91
instantiation 48, 92
local nodes 91
passing parameters 111
scope 90
Switches
simple 43
simple transistor 44
Switching Instance Data File 116

T
TAU_F 36
TAU_R 36
TDELAY 32
TH 57, 59, 65, 69, 70, 72, 73, 74, 75, 77, 79, 80, 82, 83
Thyristor 131
Time-domain Data Output 114
Title Statement 17
Transformers
ideal 43
TRIG_COND 77, 79, 80, 82, 136
TRIG_GATE 136

V
VCQNEG 45, 51, 59
VCQPOS 45, 51, 59
VCSW 51
VOFFSET 32
VOH 65, 67, 69, 70, 72, 73, 74, 75, 77, 79, 80, 82, 83
VOL 65, 67, 69, 70, 72, 73, 74, 75, 77, 79, 80, 82, 83
VPWLR 51, 53

VSAT 59

X
XOR 48, 51, 69

Copyright © Transim Technology Corporation 1991-2007
SIMPLIS Reference Manual 5/10/07